Unit 1: Velocities & Statics

Introduction to Robotics.

Robot: *'A robot is an autonomous machine capable of sensing its environment, carrying out computations to make decisions, and performing actions in the real world.'*

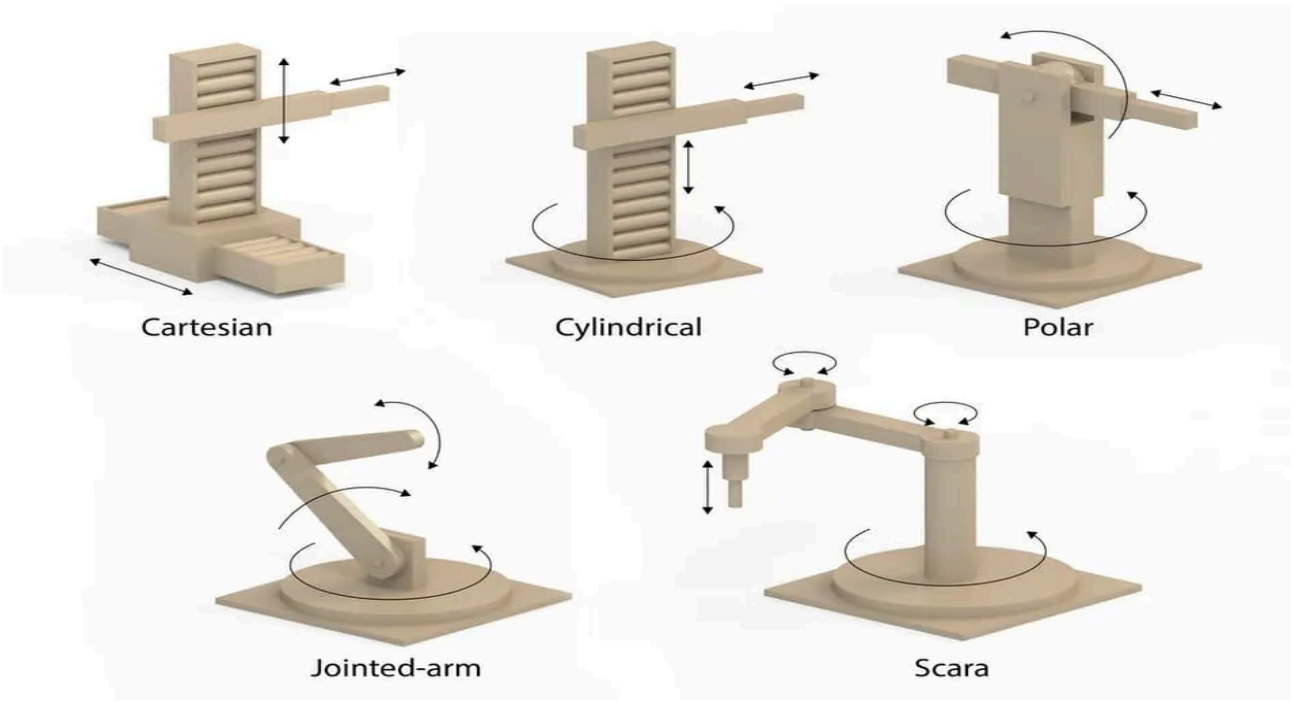
Robotics is a branch of engineering and science that includes electronics engineering, mechanical engineering and computer science and so on. This branch deals with the design, construction, use to control robots, sensory feedback and information processing. These are some technologies which will replace humans and human activities in coming years. These robots are designed to be used for any purpose but these are used in sensitive environments like bomb detection, deactivation of various bombs etc. Robots can take any form but many of them have given the human appearance. The robots which have taken the form of human appearance may likely have the walk like humans, speech, cognition and most importantly all the things a human can do. Most of the robots of today are inspired by nature and are known as bio-inspired robots. Robotics is that branch of engineering that deals with conception, design, operation, and manufacturing of robots. There was an author named Issac Asimov, he said that he was the first person to give robotics a name in a short story composed in 1940's. In that story, Issac suggested three principles were given the name of Issac's three laws of Robotics. **These three laws state that:**

a. Robots will never harm human beings.
b. Robots will follow instructions given by humans with breaking law one.
c. Robots will protect themselves without breaking other rules.

Types of Robots

- **Articulated:** The feature of this robot is its rotary joints and range of these are from 2 to 10 or more joints. The arm is connected to the rotary joint and each joint is known as the axis which provides a range of movements.
- **Cartesian:** These are also known as gantry robots. These have three joints which use the Cartesian coordinate system i.e x, y, z. These robots are provided with attached wrists to provide rotatory motion.
- **Cylindrical:** These types of robots have at least one rotatory joints and one prismatic joint which are used to connect the links. The use of rotatory joints is to rotate along the axis and prismatic joint used to provide linear motion.

- **Polar:** These are also known as spherical robots. The arm is connected to base with a twisting joint and has a combination of 2 rotatory joint and one linear joint.
- **Scara:** These robots are mainly used in assembly applications. Its arm is in cylindrical in design. It has two parallel joints which are used to provide compliance in one selected plane.



- **Delta:** The structure of these robots are spider-shaped. They are built by joint parallelograms that are connected to the common base. The parallelogram moves in a dome-shaped work area. These are mainly used in food and electrical industries.

Characteristics of robots:

- **Appearance:** Robots have a physical body. They are held by the structure of their body and are moved by their mechanical parts. Without appearance, robots will be just a software program.
- **Brain:** Another name of brain in robots is On-board control unit. Using this robot receives information and sends commands as output. With this control unit the robot knows what to do, otherwise it'll be just a remote-controlled machine.
- **Sensors:** The use of these sensors in robots is to gather info from the outside world and send it to Brain. Basically, these sensors have circuits in them that produces the voltage in them.

- **Actuators:** The robots move and the parts with the help of these robots move is called Actuators. Some examples of actuators are motors, pumps, compressors etc. The brain tells these actuators when and how to respond or move.
- **Program:** Robots only works or responds to the instructions which are provided to them in the form of a program. These programs only tell the brain when to perform which operation like when to move, produce sounds etc. These programs only tell the robot how to use sensors data to make decisions.
- **Behaviour:** Robots behavior is decided by the program which has been built for it. Once the robot starts making the movement, one can easily tell which kind of program is being installed inside the robot.

Scope and limitations of robots

The advanced version of machines are robots which are used to do advanced tasks and are programmed to make decisions on their own. When a robot is designed the most important thing to be kept in mind is that What the function is to be performed and what are the limitations of the robot. Each robot has a basic level of complexity and each of the levels has the scope which limits the functions that are to be performed. For general basic robots, their complexity is decided by the number of limbs, actuators and the sensors that are used while for advanced robots the complexity is decided by the number of microprocessors and microcontroller used. As increasing any component in the robot, it is increasing the scope of the robot and with every joint added, the degree of the robot is enhanced.

Advantages:

- They can get information that a human can't get.
- They can perform tasks without any mistakes and very efficiently and fast.
- Maximum robots are automatic, so they can perform different tasks without needing human interaction.
- Robots are used in different factories to produce items like plane, car parts etc.
- They can be used for mining purposes and can be sent to earth's madrid.
- Increased Efficiency: Robots can work 24/7 without getting tired, leading to increased productivity and efficiency.
- Improved Accuracy: Robots are capable of performing tasks with high precision and accuracy, reducing errors and improving quality.
- Increased Safety: Robots can perform tasks that are dangerous for humans, improving overall safety in the workplace.
- Reduced Labor Costs: The use of robots can lead to reduced labor costs, as robots can perform tasks more cheaply than human workers.

Disadvantages:

- They need the power supply to keep going. People working in factories may lose their jobs as robots can replace them.
- They need high maintenance to keep them working all day long. And the cost of maintaining the robots can be expensive.
- They can store huge amount of data but they are not as efficient as our human brains.
- As we know that robots work on the program that has been installed in them. So other than the program installed, robots can't do anything different.
- The most important disadvantage is that if the program of robots comes in wrong hands they can cause the huge amount of destruction.
- Initial Cost: Implementing and maintaining a robotics system can be expensive, especially for small and medium-sized businesses.
- Job Losses: The increased use of robots may result in job losses for human workers, particularly in industries where manual labor is prevalent.
- Limited Capabilities: Robots are still limited in their capabilities compared to human workers and may not be able to perform tasks requiring dexterity or creativity.
- Maintenance Costs: Robots require regular maintenance and repair, which can be time-consuming and expensive.

Applications of robots

Different types of robots can perform different types of tasks. For example, many of the robots are made for assembly work which means that they are not relevant for any other work and these types of robots are called Assembly Robots. Similarly, for seam welding many suppliers provide robots with their welding materials and these types of robots are known as Welding Robots. While on the other hand many robots are designed for heavy-duty work and are known as Heavy Duty Robots. There are some applications given below:

- A robot can also do Herding tasks.
- Robots are increasingly being used more than humans in manufacturing while in the auto-industry more than half of the labourers are "Robots".
- Many of the robots are used as Military Robots.
- Robots have been used in cleaning up areas like toxic waste or industrial wastes etc.
- Agricultural robots.
- Household robots.
- Domestic robots.
- Nano robots.

- Swarm robots.

Definition and significance of velocities in robotics

Definition:
In robotics, velocities represent the dynamic aspect of motion, quantifying the rate of change of an object's position concerning time. Mathematically, linear velocity ($v(t)$) is expressed as the derivative of the position vector ($r(t)$) with respect to time:

$$v(t) = \frac{dr(t)}{dt}$$

Examples:

1. **Automated Manufacturing Robots:** In a manufacturing setup, robot arms exhibit velocities to precisely assemble components. The linear velocity of the end effector determines how fast it moves between different positions on the assembly line.
2. **Robotics Surgery Systems:** In surgical robots, velocities dictate the movement of robotic arms during delicate procedures. The surgeon controls the linear velocity to guide the robot's tools with precision.
3. **Autonomous Vehicles:** Velocities are crucial for self-driving cars and drones. The linear velocity determines how fast they travel, while angular velocity influences their steering and orientation.
4. **Warehouse Automation:** Robots in warehouses use velocities for efficient navigation. Linear velocity helps them move swiftly between shelves, while angular velocity aids in turning and aligning with objects.
5. **Exoskeletons for Rehabilitation:** In robotic exoskeletons, velocities are vital for providing natural movements. Linear velocity determines walking speed, and angular velocity controls joint movements, ensuring a seamless gait for rehabilitation purposes.
6. **Space Exploration Robots:** In space rovers, velocities govern their movements on celestial bodies. Linear velocity directs the rover across the surface, while angular velocity adjusts its orientation for data collection.

In these examples, velocities are fundamental to the dynamic behaviour of robots, allowing them to perform diverse tasks with precision and adaptability. Whether assembling products, assisting in surgeries, or exploring new frontiers, the understanding and control of velocities are essential for the successful operation of robotic systems.

Significance:

- **Precision in Motion:** Velocities play a pivotal role in achieving precise and controlled movements of robotic systems. Understanding and manipulating velocities are crucial for tasks that demand accuracy and efficiency.

- **Task Efficiency:** Efficient control over velocities enables robots to perform tasks with increased speed and effectiveness. This is particularly essential in applications where quick and accurate responses are paramount.
- **Dynamic Adaptation:** Velocities allow robots to dynamically adapt to changing environments. The ability to adjust speed and direction in real-time is vital for robots operating in diverse and unpredictable scenarios.
- **Kinematic Planning:** Velocities are fundamental in kinematic planning, facilitating the coordination of joint movements to achieve desired end-effector motions. This is critical for applications ranging from manufacturing to advanced robotic surgeries.
- **Safety Considerations:** Understanding and controlling velocities are essential for ensuring the safety of both the robotic system and its human collaborators. Precise velocity control helps prevent collisions and ensures smooth interaction in shared workspaces.

Robotic Manipulators

Definition:

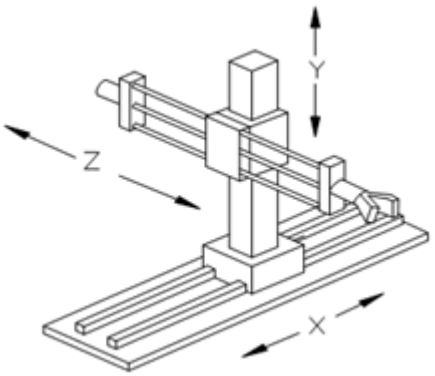
A Robot Manipulator is the arm-like structure of an industrial robot responsible for carrying out programmed tasks. It is a complex system comprising multiple links and joints, akin to the limbs of the human body, allowing it to perform a diverse range of movements within its designated workspace or work envelope.

Components:

- ***Links****:* These are rigid components that connect various sections of the robot arm, providing structural integrity.
- ***Joints****:* Essential for flexibility, joints enable different modes of motion such as linear, rotary, and revolutionary, allowing the robot to achieve the desired movements within its work envelope.

Types of Robot Manipulators:

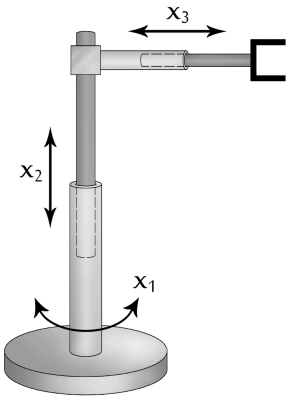
Gantry or Cartesian Robot Manipulators:



- *Description:* Mounted overhead, these manipulators have three prismatic joints, facilitating linear sliding motions.

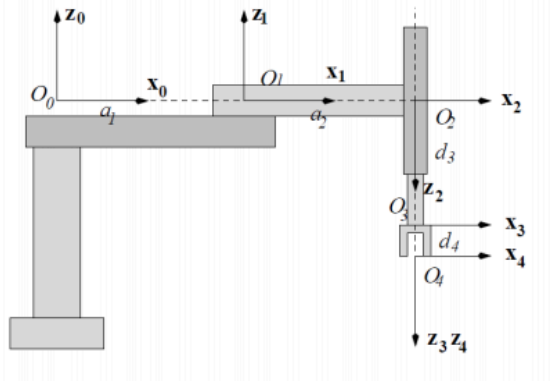
- *Characteristics:* Classified as 3-axis robots, they possess large rectangular or cubic work envelopes.

Cylindrical Robot Manipulators:



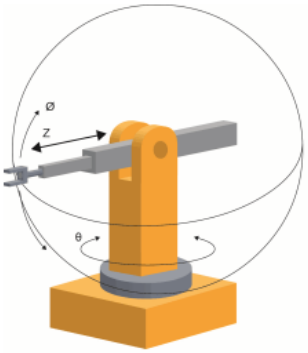
- *Description:* Named for the shape of their work envelope, these manipulators have linear joints connected to a rotary base, enabling movement along the vertical axis.
- *Characteristics:* They can elongate and retract within a 360-degree range.

Selective Compliant Assembly Robot Arm (SCARA) Manipulators:



- *Description:* Equipped with rotary joints, SCARA manipulators move along the X/Y axis while remaining rigid in the Z axis.
- *Characteristics:* Classified as 4-axis robots, they boast a wide, semi-circular work envelope.

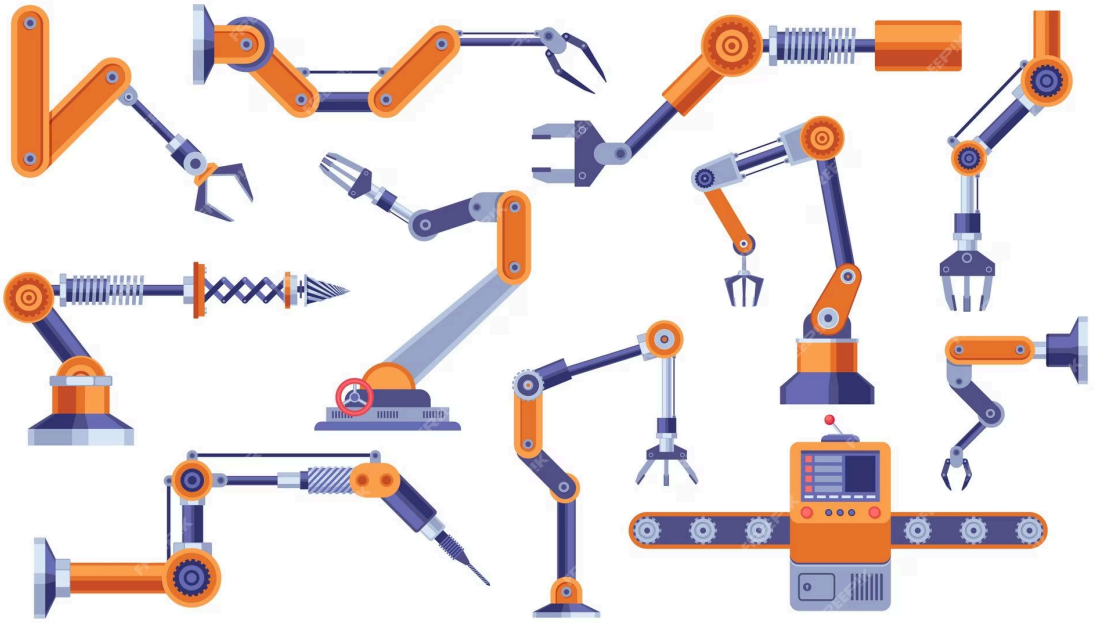
Polar Robot Manipulators:



- *Description:* Combining rotary and linear joints, polar manipulators use a revolute joint to mount to a base, enabling circular motions within three axes for a spherical work envelope.

Articulated Robot Manipulators:

- *Description:* The most flexible configuration, articulated manipulators connect to the base via a rotary joint, and links within the arm connect using revolute joints.
- *Characteristics:* Classified as 6-axis robots, they can perform complex movements like rolling, pitching, and yawing.

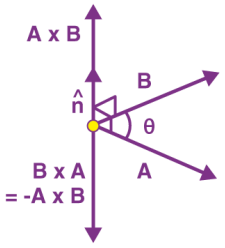


Examples of a few of many Robotic Manipulators

Introduction to the Cross Product Operator for kinematics.

In kinematics, the Cross Product Operator is a mathematical operation that combines two vectors to produce a third vector, perpendicular to the plane of the original vectors. It's symbolised by the cross product symbol (×). In the context of robotics, this operator is particularly valuable for understanding the interplay between linear and angular velocities.

The vector product or cross product of two vectors A and B is denoted by A × B, and its resultant vector is perpendicular to the vectors A and B. The cross product is mostly used to determine the vector, which is perpendicular to the plane surface spanned by two vectors, whereas the dot product is used to find the angle between two vectors or the length of the vector. The cross product of two vectors, say A × B, is equal to another vector at right angles to both, and it happens in the three dimensions.



Key points to remember:

- The cross product of two vectors is always a vector quantity.
- In vector product, the resulting vector contains a negative sign if the order of vectors is changed.
- The direction of $\vec{A} \times \vec{B}$ is always perpendicular to the plane containing $\vec{A}$ and $\vec{B}$.
- The cross product of any two linear vectors is always a null vector.

If θ is the angle between the given two vectors A and B, then the formula for the cross product of two vectors using the determinant of the matrix is as given below.

A = ai + bj + ck B = xi + yj + zk

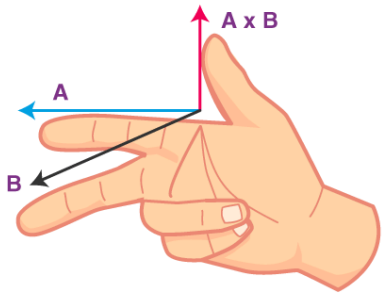
$$A \times B = \begin{vmatrix} i & j & k \\ a & b & c \\ x & y & z \end{vmatrix}$$

A X B = (bz – cy)i – (az – cx)j + (ay – bx)k = (bz – cy)i + (cx – az)j + (ay – bx)k

Right-hand Rule Cross Product: We can find the direction of the unit vector with the help of the right-hand rule.

In this rule, we can stretch our right hand so that the index finger of the right hand is in the direction of the first vector and the middle finger is in the direction of the second vector. Then, the thumb of the right hand indicates the direction or unit vector n. With the help of the

right-hand rule, we can easily show that vectors' cross product is not commutative. If we have two vectors A and B, then the diagram for the right-hand rule is as follows:



To find the cross product of two vectors, we can use properties. The properties such as anti-commutative property, zero vector property plays an essential role in finding the cross product of two vectors. Apart from these properties, some other properties include Jacobi property, distributive property. The properties of cross-product are given below:

1. Anti-commutative Property : $\vec{A} \times \vec{B} = -\vec{B} \times \vec{A}$
2. Distributive Property : $\vec{A} \times (\vec{B} + \vec{C}) = \vec{A} \times \vec{B} + \vec{A} \times \vec{C}$
3. Jacobi Property: $\vec{A} \times (\vec{B} \times \vec{C}) + \vec{B} \times (\vec{C} \times \vec{A}) + \vec{C} \times (\vec{A} \times \vec{B}) = 0$
4. Zero Vector Property: $a \times b = 0$ if a = 0 or b =0.

Cross Product Example

Example:

Find the cross product of the given two vectors:

$\vec{X} = 5\vec{i} + 6\vec{j} + 2\vec{k}$ *and* $\vec{Y} = \vec{i} + \vec{j} + \vec{k}$

Solution:

Given:

$\vec{X} = 5\vec{i} + 6\vec{j} + 2\vec{k}$

$\vec{Y} = \vec{i} + \vec{j} + \vec{k}$

To find the cross product of two vectors, we have to write the given vectors in determinant form. Using the determinant form, we can find the cross product of two vectors as:

$$\vec{X} \times \vec{Y} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ 5 & 6 & 2 \\ 1 & 1 & 1 \end{vmatrix}$$

By expanding,

$\vec{X} \times \vec{Y} = (6-2)\vec{i} - (5-2)\vec{j} + (5-6)\vec{k}$

Therefore,

$\vec{X} \times \vec{Y} = 4\vec{i} - 3\vec{j} - \vec{k}$

Jacobian in a frame

In robotics, the Jacobian matrix holds a crucial role within **frame manipulation** and **kinematics**, specifically related to analyzing the relationship between joint velocities and end-effector velocities. Here's a breakdown:

Understanding the Frame:

Robots operate in different **coordinate frames**, which serve as reference points for describing positions and orientations. These frames can be associated with the base, individual links, or the end-effector. Choosing the appropriate frame is crucial for analyzing robot motion.

What is the Jacobian?

The Jacobian, denoted by **J**, is a rectangular matrix that acts as a **mapping tool**. It translates the **velocities of each joint** (represented as a vector) in the robot's structure into the **linear and angular velocities** of the end-effector (also represented as a vector). In essence, it tells you how much each joint's movement contributes to the overall motion of the end-effector.

Mathematically:

The Jacobian relates joint velocities ($\dot{q}$) to end-effector velocities ($\dot{v}$) through the equation:

$$\dot{v} = J * \dot{q}$$

where:

- $\dot{v}$ is a 6x1 vector containing the linear and angular velocities of the end-effector (3 linear, 3 angular)
- $\dot{q}$ is an nx1 vector containing the velocities of all n joints
- J is an mxn Jacobian matrix, where m is the number of end-effector velocities (usually 6) and n is the number of joints

Why is it important?

The Jacobian is invaluable for various tasks in robotics:

- **Inverse Kinematics:** Given a desired end-effector pose (position and orientation), the Jacobian helps **calculate the required joint angles** to achieve it.
- **Path Planning and Trajectory Generation:** By understanding how joint movements translate to end-effector motion through the Jacobian, robots can plan smooth and efficient paths.
- **Control and Simulation:** Control algorithms utilize the Jacobian to translate desired end-effector velocities into specific joint velocity commands. Simulating robot movements also relies on accurate Jacobian calculations.

Key points to remember:

- The Jacobian depends on the chosen frame for the end-effector.
- It's not always square, reflecting different numbers of joints and possible end-effector velocities.
- Understanding the Jacobian is crucial for analyzing, controlling, and simulating robot motion in various applications.

Additional aspects:

- In complex robots with many joints, the Jacobian calculation might be more intricate.
- Singularities occur when the Jacobian loses rank, meaning certain joint movements won't affect the end-effector in specific configurations. This is crucial to avoid during robot operation

Kinematic Singularity (When Movement Stalls)

Kinematic Singularity in Robotics: Kinematic singularities are the concepts in robotics, representing configurations where the robot loses some or all of its **degrees of freedom**. In simpler terms, it's like hitting a "dead end" where joint movements no longer translate to predictable end-effector motion. Let's delve deeper:

What happens during a singularity?

Think of a robotic arm trying to reach a point. Normally, each joint movement contributes to the arm's extension and direction. But at a singularity, one or more joint motions no longer affect the end-effector's position or orientation, even though the joints themselves might be moving. This can manifest in various ways:

- **Loss of control:** The robot loses the ability to precisely control the end-effector's pose (position and orientation) in certain directions.
- **Infinite joint velocities:** In rare cases, joint velocities might need to become infinitely large to achieve a small change in the end-effector.
- **Indeterminacy:** The mapping between joint angles and end-effector pose becomes unclear, leading to unexpected or unpredictable behavior.

Identifying Singularity Types:

- **Velocity singularities:** Occur when the Jacobian matrix, which maps joint velocities to end-effector velocities, loses rank. This indicates specific joint combinations won't affect the end-effector at that configuration.
- **Configuration singularities:** Arise when the robot reaches a specific joint configuration where multiple solutions exist for a given end-effector pose, potentially creating ambiguity in control.

Mathematical Representations:

- **Jacobian matrix and singularity:**
    - Recall the relationship between joint velocities ($\dot{q}$) and end-effector velocities ($\dot{v}$):

$$\dot{v} = J * \dot{q}$$

    - When det(J) = 0, the determinant of the Jacobian becomes zero, indicating a **velocity singularity**.
- **Singularity locus:** Mathematically represented as a curve or surface in the robot's joint space, describing all joint configurations where singularities occur. Analyzing this locus helps avoid these problematic areas during robot operation.

Why are singularities important?

Avoiding singularities is critical for safe and efficient robot operation:

- **Prevents loss of control:** By recognizing and avoiding singularities, you ensure predictable and controllable robot behavior.
- **Protects against damage:** Reaching a singularity can create excessive forces and stresses on robot joints, potentially leading to damage.
- **Improves performance:** Understanding and managing singularities optimizes motion planning and control algorithms, enhancing robot performance.

Real-world examples:

- A robotic arm trying to reach behind itself might encounter a singularity, losing dexterity in certain directions.
- A 6-legged robot walking over uneven terrain could face singularities where specific leg configurations limit its ability to maintain balance.

Thus, Kinematic singularities are essential considerations in robotics design, control, and operation. Understanding their mathematical underpinnings and utilizing tools like the Jacobian matrix allow us to identify and avoid these problematic configurations, ensuring smooth, controlled, and safe robot motion.

Unit 2: Robot Kinematics

Manipulator Kinematics

Manipulator kinematics is a fundamental area of robotics that deals with the **relationship between the joint configurations of a robotic manipulator and its end-effector position and orientation in space**. In simpler terms, it helps us understand how the movement of individual joints in a robot arm translates into the movement of the tool or gripper at the end of the arm (the end-effector).

There are two main aspects of manipulator kinematics:

1. Forward kinematics: This problem takes the **joint angles** of the manipulator as input and calculates the **position and orientation** of the end-effector in space. Imagine you know how far each joint of the arm has moved, and you want to know where the robot's "hand" is now.

2. Inverse kinematics: This problem takes the desired **position and orientation** of the end-effector as input and calculates the necessary **joint angles** to achieve that position. This is like setting a specific goal for the robot's hand and figuring out how each joint needs to move to reach it.

Both forward and inverse kinematics are crucial for various robotics applications, including:

- **Robot control:** To move the robot to specific positions and orientations, controllers need to solve the inverse kinematics problem to determine the required joint angles.
- **Path planning:** Planning the movement of the robot through a desired path involves solving the forward kinematics to understand how each point on the path translates to joint movements.
- **Simulation and analysis:** Simulating robot motion and analyzing its performance often rely on accurate solutions to both forward and inverse kinematics problems.

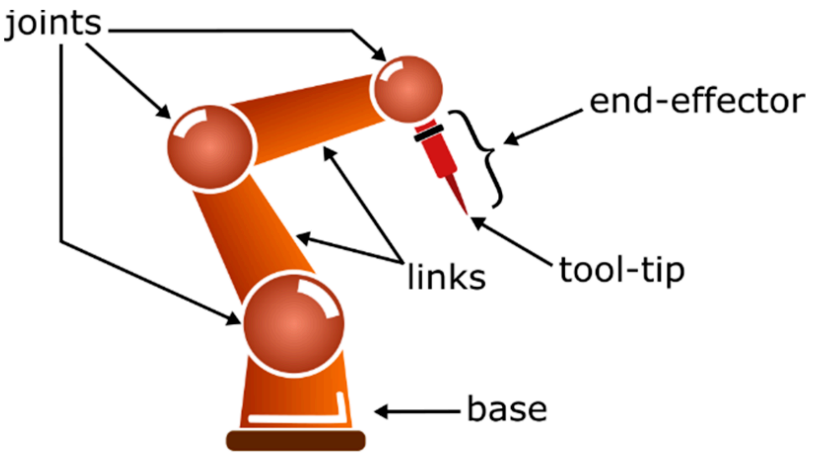
Manipulator kinematics can be solved using various methods depending on the complexity of the robot and the specific problem. These methods might involve:

- **Denavit-Hartenberg parameters:** A standardized way to describe the geometry of a robot, allowing for systematic kinematic analysis.

- **Homogeneous transformations:** Representing positions and orientations using 4x4 matrices, enabling efficient calculations.
- **Numerical methods:** Solving complex kinematics problems computationally.

Link Description

In robotics, link descriptions refer to the parameters used to define the rigid segments that connect the joints of a manipulator. These parameters are crucial for understanding the robot's geometry and performing kinematic and dynamic analysis.



Link description typically includes:

1. Link length: This is the fixed distance between the centers of two adjacent joints connected by the link. Units are typically in meters or millimeters.

2. Link offset: This represents the distance between the joint axis and the center of mass of the link. It can be considered the "thickness" of the link along its length. Units are also in meters or millimeters.

3. Joint axis: This defines the direction about which the link can rotate relative to the previous link. It's usually represented by a unit vector.

4. Denavit-Hartenberg (DH) parameters: This is a standardized method for defining link parameters using four parameters per link:

- **α (alpha):** Angle between the z-axis of the previous link and the common perpendicular line between the z-axes of the current and previous links.
- **a (a)**: Distance between the z-axes of the previous and current links along the common perpendicular line.
- **d (d)**: Offset distance between the joint axis and the common perpendicular line.

- **θ (theta):** Angle between the x-axes of the current and previous links about the z-axis of the previous link.

Additional parameters:

- **Link mass and inertia:** This information is needed for dynamic analysis to understand how forces and torques affect the robot's motion.
- **Collision geometry:** This describes the shape of the link for collision detection and planning safe robot movements.

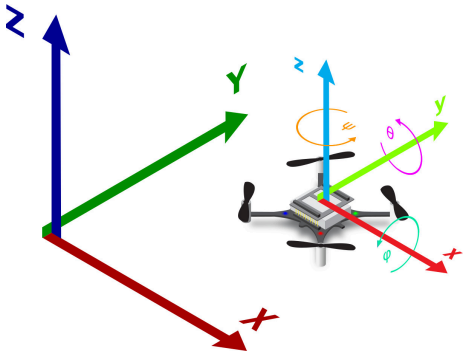
Importance of link descriptions:

- Accurate link descriptions are essential for performing **kinematic analysis**, which determines the position and orientation of the end-effector based on the joint angles.
- They are also crucial for **dynamic analysis**, which studies the forces and torques acting on the robot and its response to those forces.
- Link descriptions form the basis for robot **simulation** and **control algorithms**.

Remember, the specific link parameters and representation methods may vary depending on the robot's design and the chosen analysis tools. Understanding these descriptions is fundamental for designing, controlling, and analyzing robotic systems effectively.

Link to reference Frame Connections:

In robot kinematics, understanding the connection between links and reference frames is crucial for analyzing and controlling the robot's motion. Each link can be associated with a unique reference frame, allowing us to track its position and orientation relative to other links and the overall robot structure.



Reference Frames:

- Imagine placing a set of three orthogonal axes (x, y, z) at various points on the robot. These axes define reference frames, providing a "coordinate system" to describe positions and orientations within the robot's structure.
- We typically define a fixed base frame attached to the robot's base (foundation), and additional frames attached to each joint axis or the center of each link.
- The choice of frame placement depends on the robot's design and the desired analysis level.

Link Connection:

- Each link is associated with a reference frame attached to it. This frame moves along with the link as the robot joints rotate.
- The transformation between two connected link frames describes the relative position and orientation of one link with respect to the other. This transformation can be represented using:
    - **Homogeneous transformation matrices:** A 4x4 matrix encoding the translation and rotation between frames.
    - **Denavit-Hartenberg (DH) parameters:** A standard set of four parameters per link (α, a, d, θ) defining the relative transformation.

Importance in Kinematics:

- **Forward kinematics:** Knowing the link connections and transformations allows us to calculate the end-effector position and orientation given the joint angles (joint space to end-effector space).
- **Inverse kinematics:** Given a desired end-effector pose, we can solve for the required joint angles using the link connections and transformations (end-effector space to joint space).
- **Path planning:** Planning robot movement through a desired path involves understanding how joint movement translates to end-effector motion, relying on link connections.

Examples:

- In a robotic arm, each link frame describes the position and orientation of the arm segment relative to the previous one.
- In a mobile robot, the base frame describes the robot's position and orientation on the ground, while other frames might be attached to wheels or sensor locations.

Key Points:

- The link-to-reference frame connection is fundamental for understanding robot motion in different reference spaces.
- Different representations (homogeneous matrices, DH parameters) offer options for defining and calculating transformations.

- Accurately describing these connections is crucial for solving forward and inverse kinematics problems, path planning, and robot control.

Revolute and Prismatic Joints

In the world of robotics, movement comes from **joints**, similar to how our own bodies hinge and slide to perform various actions. Two fundamental types of joints are crucial for robot mobility: **revolute joints** and **prismatic joints**. Understanding their characteristics and differences is essential for designing, controlling, and analyzing robotic systems.

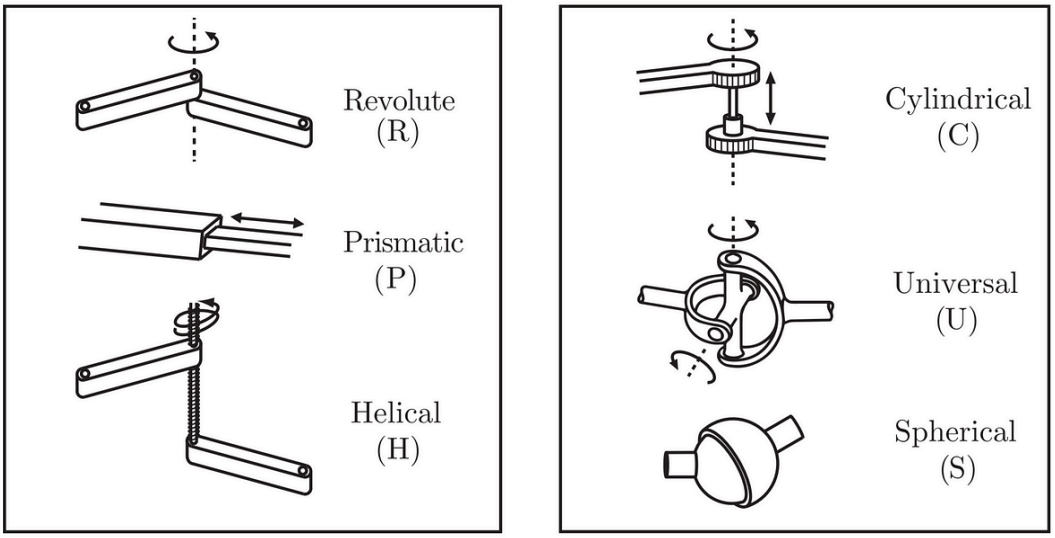


Image: Types of Joints

1. Revolute Joint:

Imagine a door hinge. That's the essence of a revolute joint, allowing **rotation** about a single fixed axis. It's like a rotating platform, letting a robot arm segment swing or twist. Key features:

- **One degree of freedom:** Only rotation around the designated axis is possible, providing control over angular movement.
- **Common applications:** Robotic arms, legs, joints connecting segments, and manipulators with rotational capabilities.
- **Representation:** Often modeled using the angle of rotation (θ) about the joint axis.

2. Prismatic Joint:

Picture a drawer sliding out smoothly. That's the principle of a prismatic joint, enabling **linear translation** along a straight line. It's like a sliding mechanism, allowing a robot arm segment to extend or retract. Key features:

- **One degree of freedom:** Only linear movement along the fixed direction is possible, providing control over position along the axis.
- **Common applications:** Robotic legs, linear actuators, robot platforms for translational movement, and manipulators requiring linear positioning.
- **Representation:** Often modeled using the distance (d) traveled along the joint axis.

Differentiating the two:

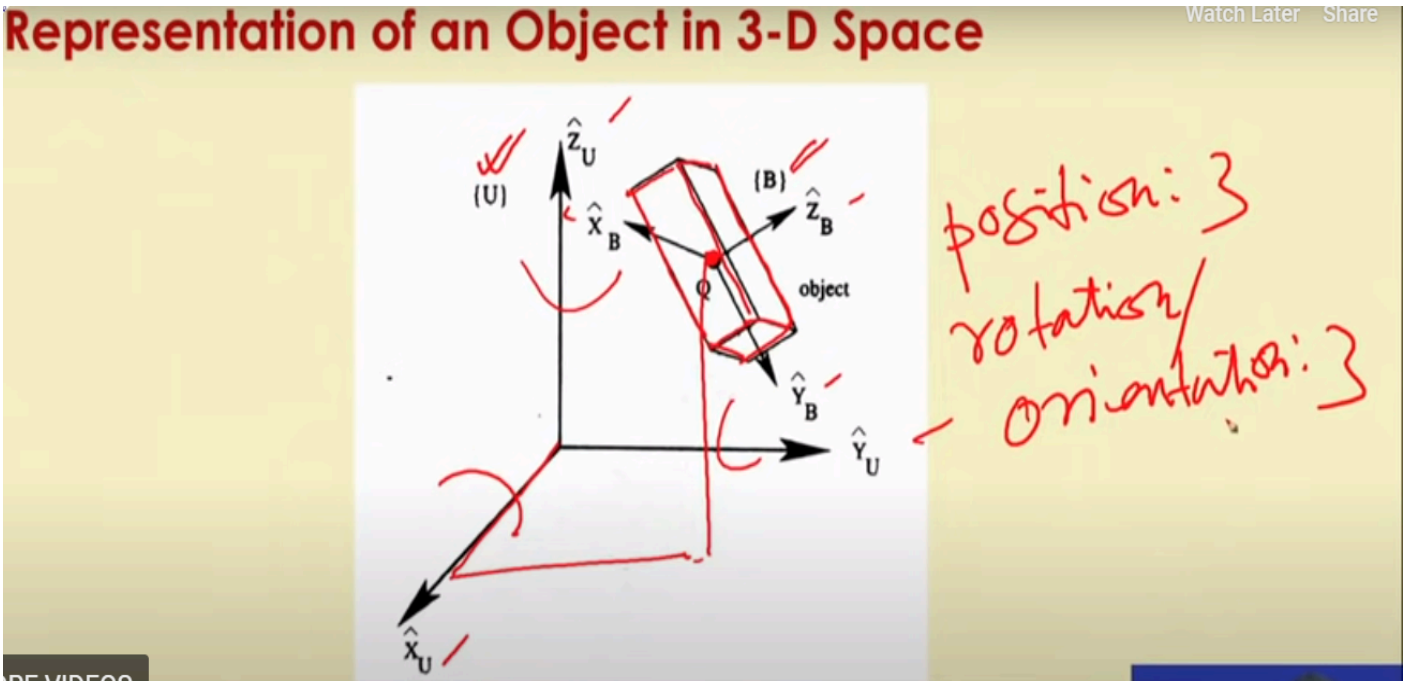
Feature	Revolute Joint	Prismatic Joint
Movement Type	Rotation around a fixed axis	Linear translation along a fixed axis
Degrees of Freedom (DOF)	1 (angular)	1 (linear)
Motion Example	Door hinge, robotic arm joints	Drawer sliding, robot platform movement
Symbol	Often represented by θ (angle)	Often represented by d (distance)
Joint Representation	Rotation about an axis represented by a unit vector and an angle	Translation along a vector representing the joint axis
Common Applications	Manipulators with rotational capabilities, robotic arms and legs, joints connecting segments	Linear actuators, robotic legs, robot platforms for translational movement, reaching and positioning
Kinematic Analysis	Manipulators with rotational capabilities, robotic arms and legs, joints connecting segments	Requires modeling translation vectors using joint displacements (d)
Limitations	Rotational movement restricted to a single axis	Cannot achieve rotational movements
Advantages	Simple to design and control	Offers precise linear positioning
Disadvantages	Limited movement complexity	Cannot manipulate objects requiring rotation
Real-world Examples	Robotic arm joints, door hinges, robot arm grippers	Robot platforms, linear actuators, drawer slides

Remember:

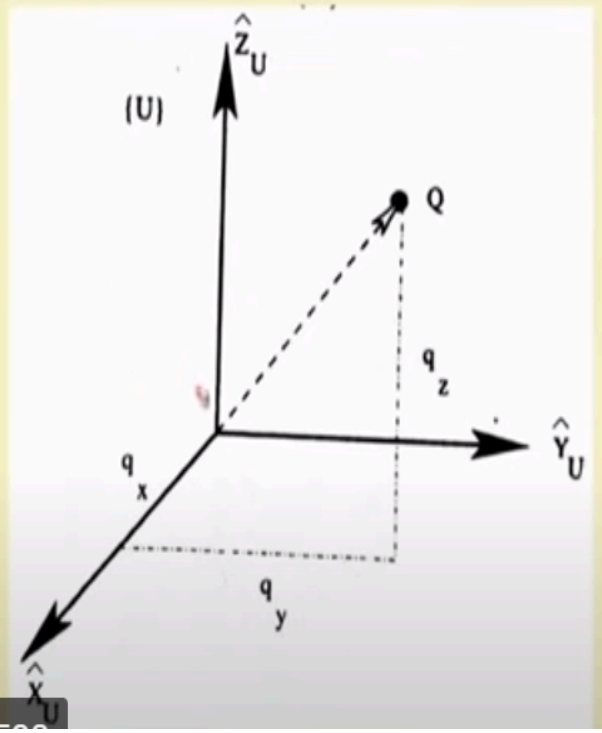
- Robots can combine these joints to achieve complex movements. For example, a robotic arm might have both revolute and prismatic joints for reaching, grasping, and manipulating objects.
- Choosing the right joint type depends on the desired robot function and movement goals.

- Kinematic analysis, which deals with robot positions and orientations based on joint states, relies heavily on understanding these joint types and their mathematical representations.

Representation of an Object in 3D Space (Homogeneous Transformation Matrix)

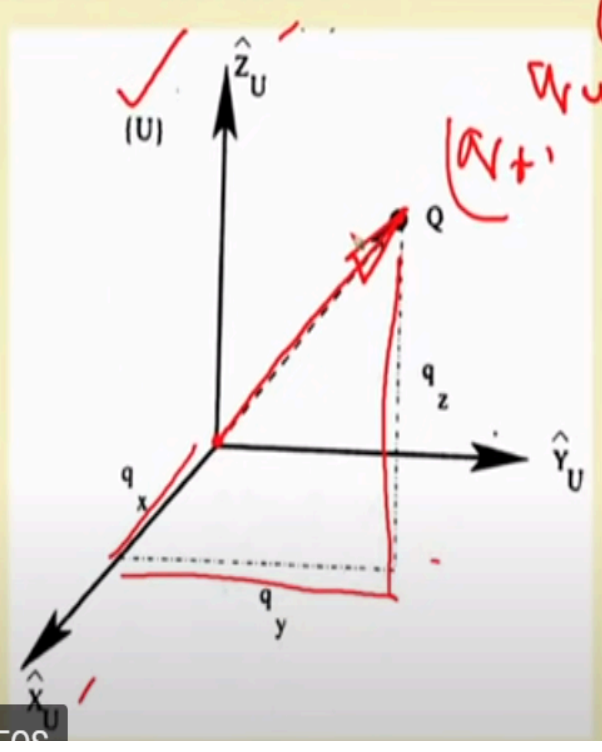


Representation of the Position



$$^{U}\overline{Q} = \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix} \text{ ; 3x1 matrix}$$

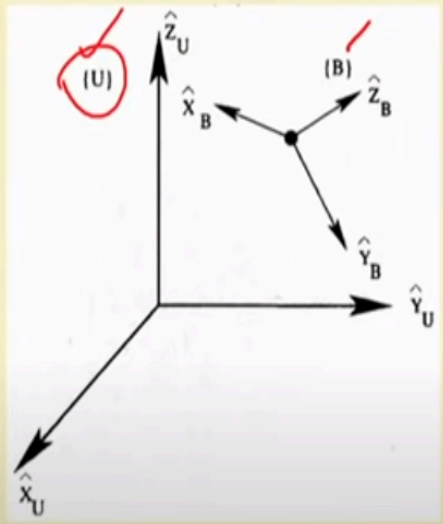
Representation of the Position



$$^{U}\overline{Q} = \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix} \text{ ; 3x1 matrix}$$

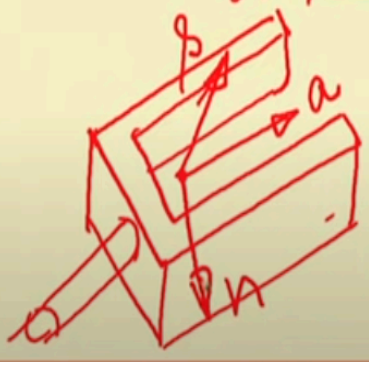
Representation of the Orientation

Cartesian coord.



$${}^U_B R = [{}^U\hat{X}_B \; {}^U\hat{Y}_B \; {}^U\hat{Z}_B]_{3\times3 \text{ matrix}}$$

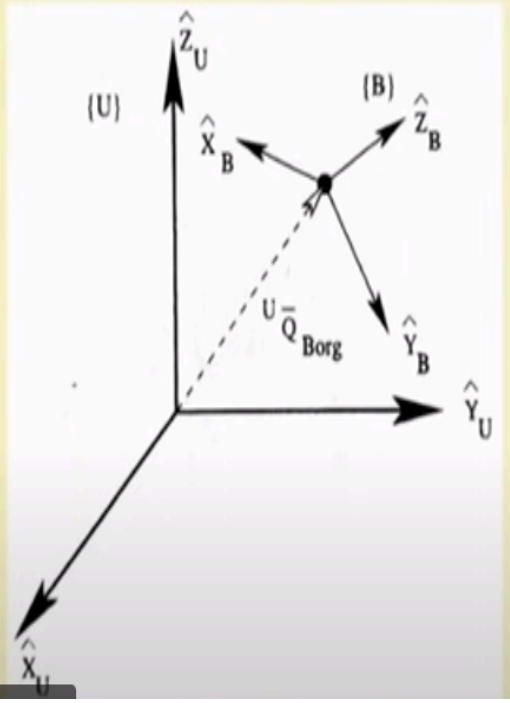
a set of 3 vector



$$\begin{bmatrix} n_x & s_x & a_x \\ n_y & s_y & a_y \\ n_z & s_z & a_z \end{bmatrix}$$

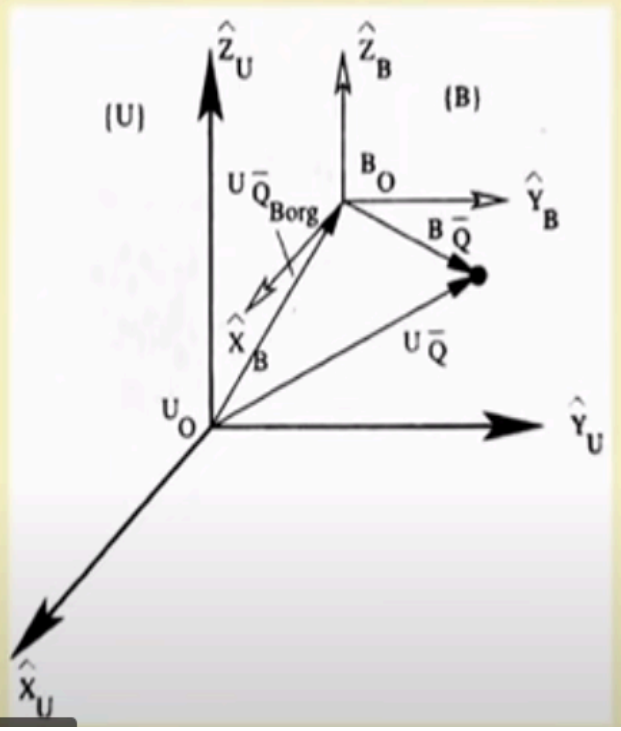
ORE VIDEOS

Frame Transformations



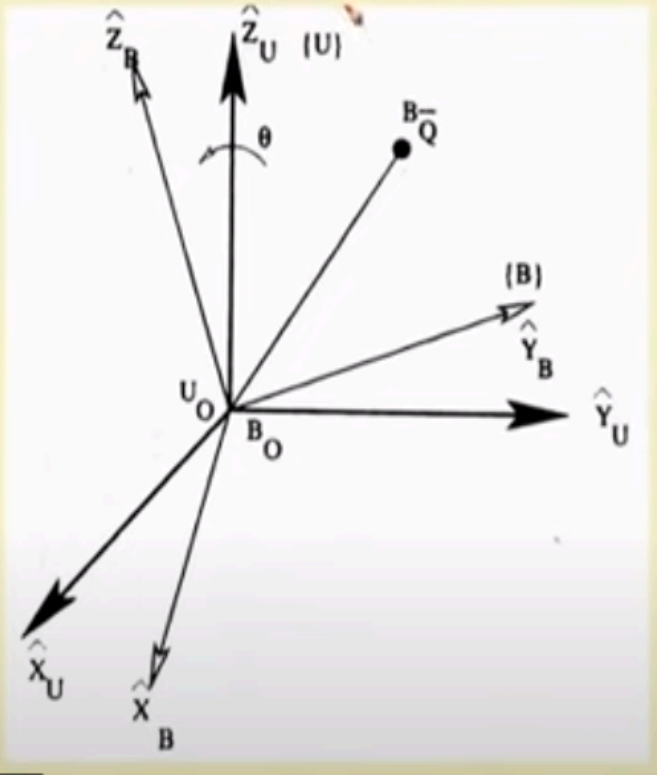
Frame: A set of four vectors carrying position and orientation information

Translation of a Frame



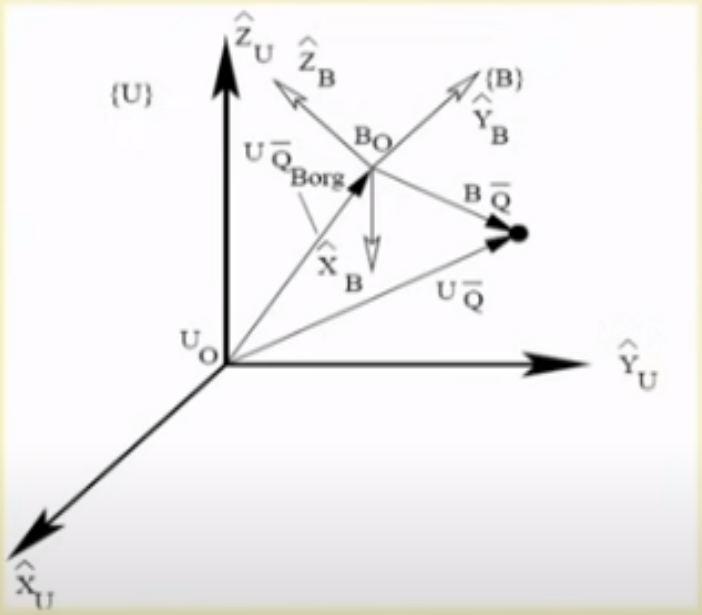
$$^{U}\overline{Q} = {}^{U}\overline{Q}_{Borg} + {}^{B}\overline{Q}$$

Rotation of a Frame



$$^{U}\overline{Q} = {}^{U}_{B}R\,{}^{B}\overline{Q}$$

Translation and Rotation of a Frame



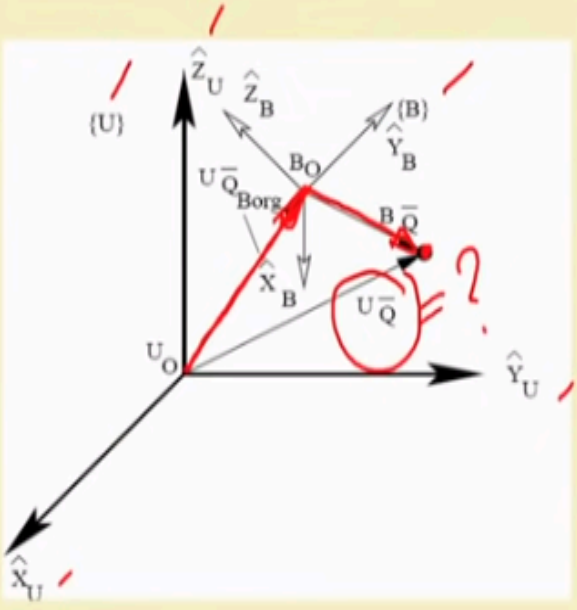
$$ {}^U\overline{Q} = {}^U_B R\, {}^B\overline{Q} + {}^U\overline{Q}_{Borg} $$

$$ {}^U\overline{Q} = {}^U_B R\, {}^B\overline{Q} + {}^U\overline{Q}_{Borg} $$

$$ \Rightarrow {}^U\overline{Q} = {}^U_B T\, {}^B\overline{Q} $$

where T: transformation

Translation and Rotation of a Frame

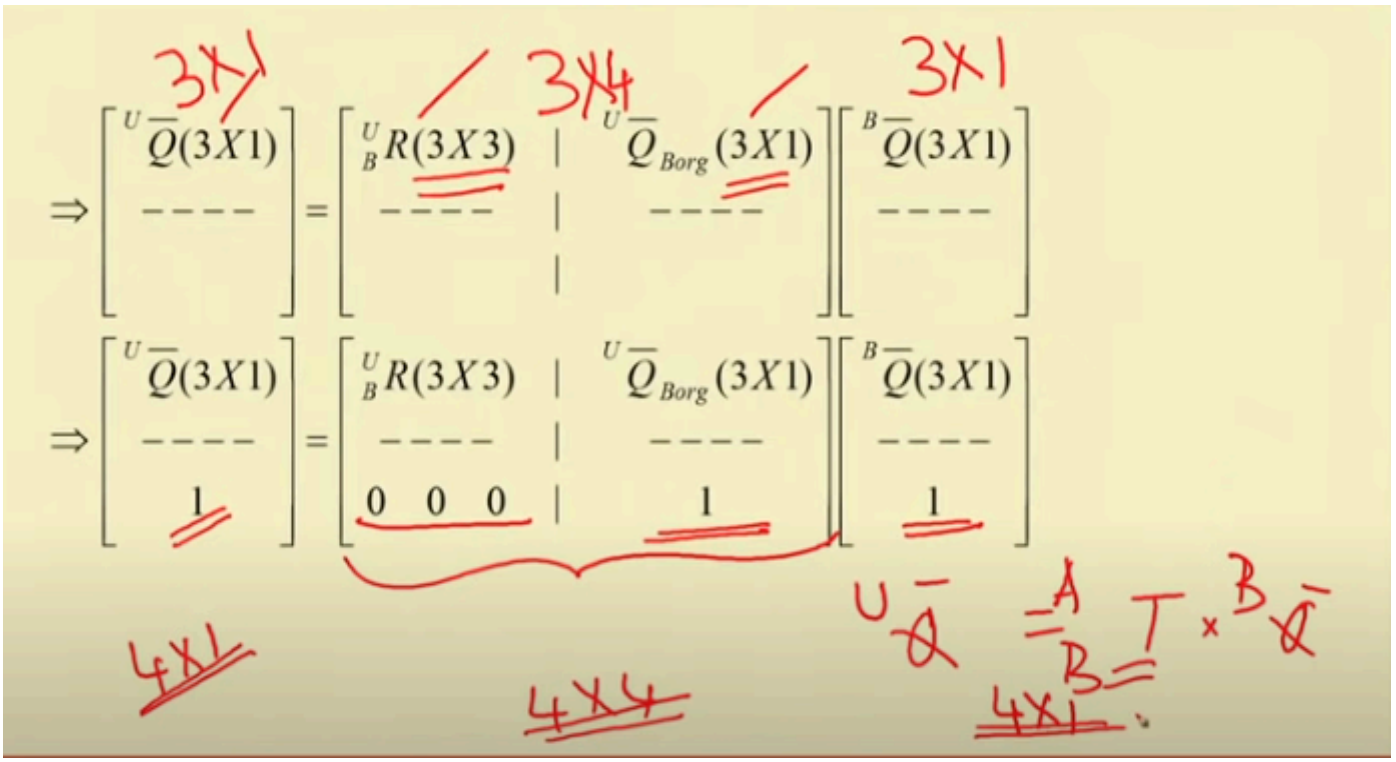


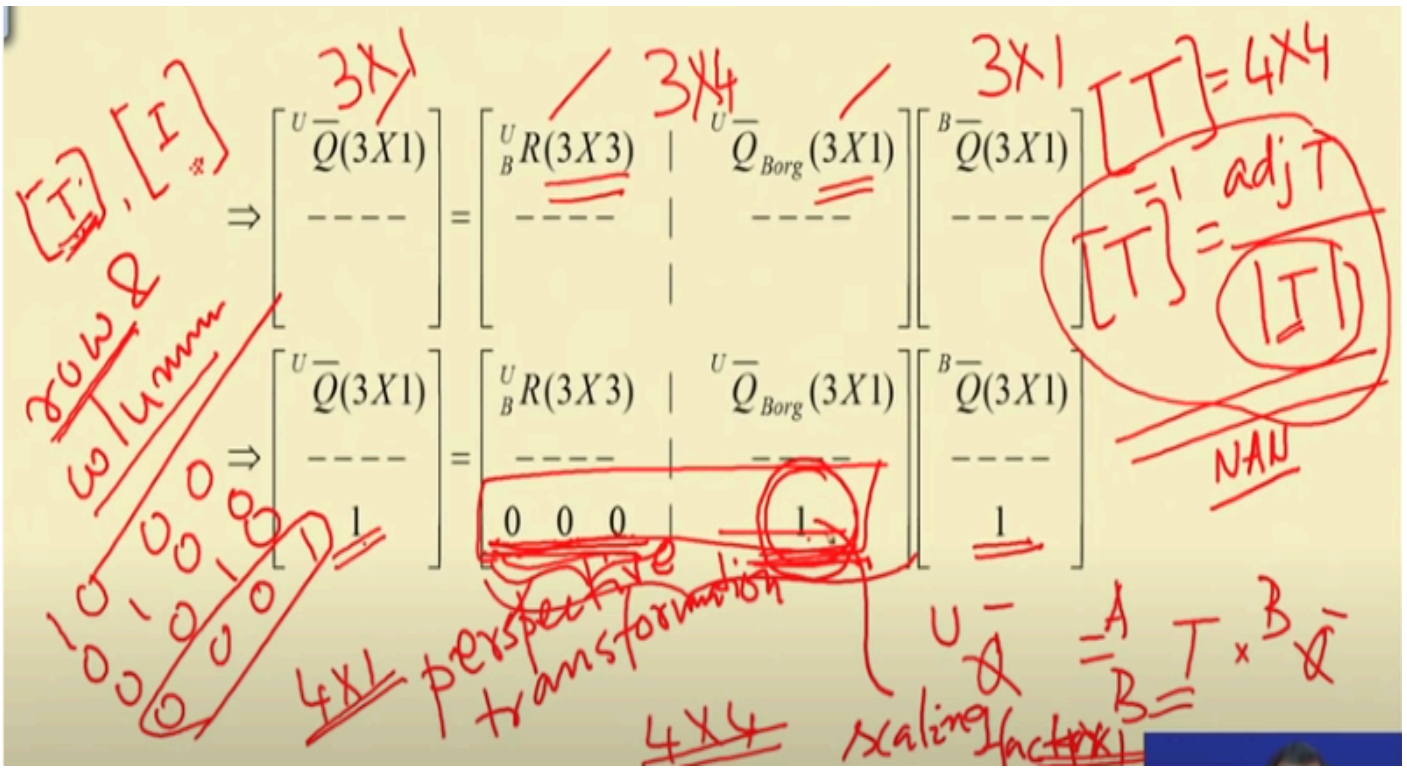
$$ {}^U\overline{Q} = {}^U_B R\, {}^B\overline{Q} + {}^U\overline{Q}_{Borg} $$

$$ {}^U\overline{Q} = {}^U_B R\, {}^B\overline{Q} + {}^U\overline{Q}_{Borg} $$

$$ \Rightarrow {}^U\overline{Q} = {}^U_B T\, {}^B\overline{Q} $$

where T: transformation = translation & rotation


3X1
3X4
3X1
$\Rightarrow \begin{bmatrix} {}^{U}\bar{Q}(3X1) \\ \hline \end{bmatrix} = \begin{bmatrix} {}^{U}_{B}R(3X3) & | & {}^{U}\bar{Q}_{Borg}(3X1) \\ \hline & | & \end{bmatrix} \begin{bmatrix} {}^{B}\bar{Q}(3X1) \\ \hline \end{bmatrix}$
$\Rightarrow \begin{bmatrix} {}^{U}\bar{Q}(3X1) \\ \hline 1 \end{bmatrix} = \begin{bmatrix} {}^{U}_{B}R(3X3) & | & {}^{U}\bar{Q}_{Borg}(3X1) \\ \hline 0\ 0\ 0 & | & 1 \end{bmatrix} \begin{bmatrix} {}^{B}\bar{Q}(3X1) \\ \hline 1 \end{bmatrix}$
4X1
4X4
4X1
${}^{U}\bar{Q} = {}^{A}_{B}T \times {}^{B}\bar{Q}$


3X1
3X4
3X1
$[T] = 4X4$
$[T]^{-1} = \frac{adj\,T}{|T|}$
NAN
$\Rightarrow \begin{bmatrix} {}^{U}\bar{Q}(3X1) \\ \hline \end{bmatrix} = \begin{bmatrix} {}^{U}_{B}R(3X3) & | & {}^{U}\bar{Q}_{Borg}(3X1) \\ \hline & | & \end{bmatrix} \begin{bmatrix} {}^{B}\bar{Q}(3X1) \\ \hline \end{bmatrix}$
$\Rightarrow \begin{bmatrix} {}^{U}\bar{Q}(3X1) \\ \hline 1 \end{bmatrix} = \begin{bmatrix} {}^{U}_{B}R(3X3) & | & {}^{U}\bar{Q}_{Borg}(3X1) \\ \hline 0\ 0\ 0 & | & 1 \end{bmatrix} \begin{bmatrix} {}^{B}\bar{Q}(3X1) \\ \hline 1 \end{bmatrix}$
[I], [I]
row & column
4X1
perspective transformation
4X4
scaling factor
${}^{U}\bar{Q} = {}^{A}_{B}T \times {}^{B}\bar{Q}$

Let [T]: Homogeneous transformation matrix

$$[T]=\left[\begin{array}{c|c} {}^{U}_{B}R(3x3) & {}^{U}\bar{Q}_{Borg}(3x1) \\ \hline 0\ \ 0\ \ 0 & 1 \end{array}\right]$$

4X4

Say

$$[T]=\begin{bmatrix} r_{11} & r_{12} & r_{13} & q_x \\ r_{21} & r_{22} & r_{23} & q_y \\ r_{31} & r_{32} & r_{33} & q_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation Operator

Trans $(\hat{X}, q)$: Translation of q units along x-direction

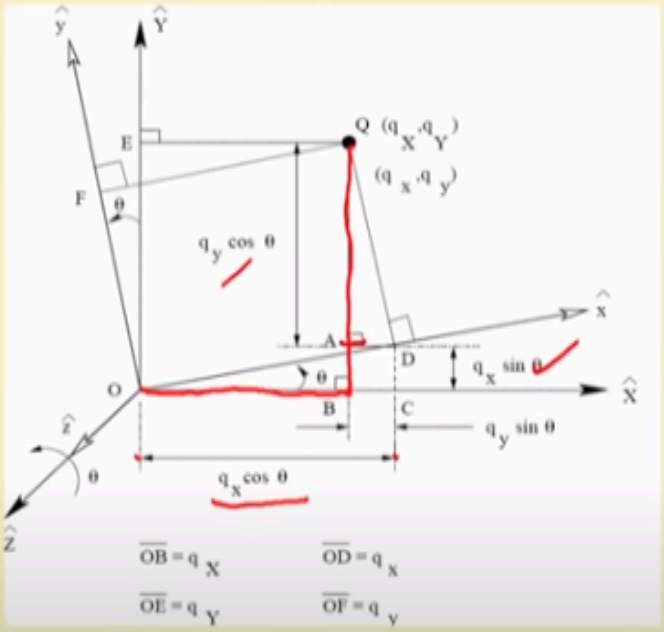
Trans $(\hat{X}, q)$ =

$$\begin{bmatrix} 1 & 0 & 0 & q \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Trans (a, b, c) = Trans $(\hat{X}, a)$ Trans $(\hat{Y}, b)$ Trans $(\hat{Z}, c)$

Note: Trans operators are commutative in nature

$$Trans(\hat{X}, q_x)\,Trans(\hat{Y}, q_y) = Trans(\hat{Y}, q_y)\,Trans(\hat{X}, q_x)$$



Rot $(\hat{Z}, \theta)$ = ?

$$q_X = q_x Cos\theta - q_y Sin\theta + q_z X0$$

$$q_Y = q_x Sin\theta + q_y Cos\theta + q_z X0$$

$$q_Z = q_x X0 + q_y X0 + q_z X1$$

In matrix form:

$$\begin{bmatrix} q_X \\ q_Y \\ q_Z \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix}$$

$$\mathrm{Rot}(\hat{Z},\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Similarly, we get

$$\mathrm{Rot}(\hat{X},\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$\mathrm{Rot}(\hat{Y},\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

Properties of Rotation Matrix

- Each row/column of a rotation matrix is a unit vector
- Inner (dot) product of each row of a rotation matrix with each other row becomes equal to 0. The same is true for each column also.
- Rotation matrices are not commutative in nature

$$ROT(\hat{X},\theta_1)ROT(\hat{Y},\theta_2) \neq ROT(\hat{Y},\theta_2)ROT(\hat{X},\theta_1)$$

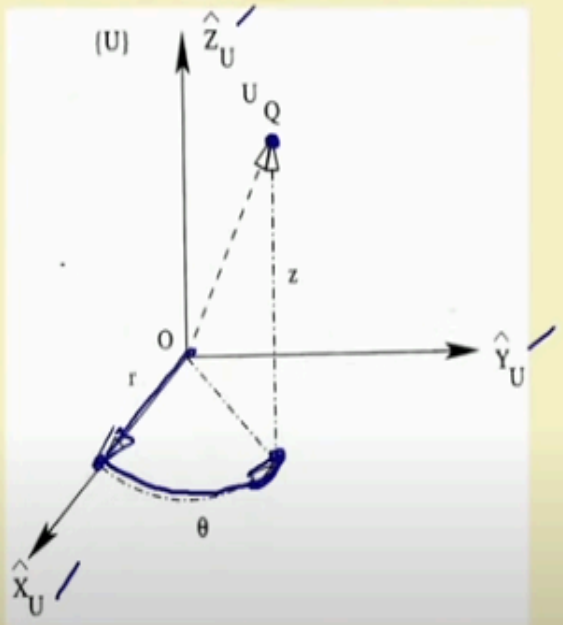
- Inverse of a rotation matrix is nothing but its transpose

$$ROT^{-1}(\hat{X},\theta) = ROT^{T}(\hat{X},\theta)$$

- ${}^{A}_{B}T = {}^{B}_{A}T^{-1}$

Cylindrical & Spherical Coordinate Systems

Cylindrical Coordinate System



Steps:

1. Starting from the origin O, translate by r units along $\hat{X}_U$ axis
2. Rotate in anti-clockwise sense about $\hat{Z}_U$ axis by an angle θ
3. Translate along $\hat{Z}_U$ axis by z units

$$[T]_{composite} = TRANS(\hat{Z}_U, z)\, ROT(\hat{Z}_U, \theta)\, TRANS(\hat{X}_U, r)$$

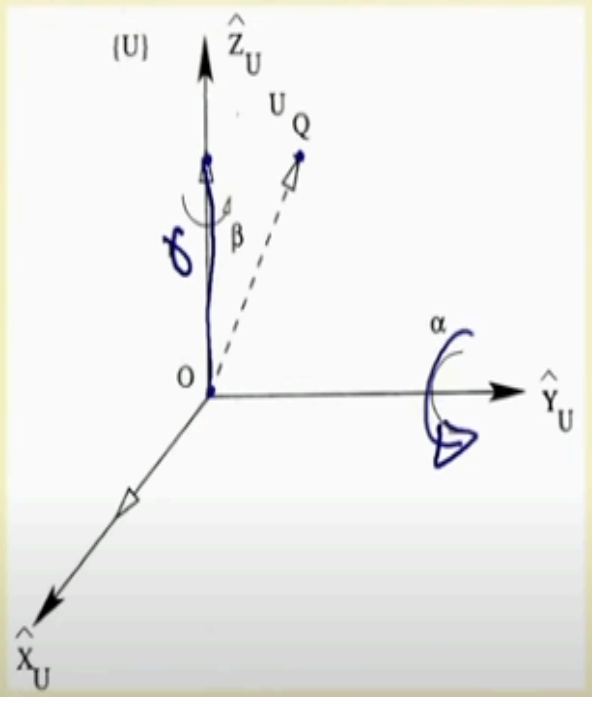
$$= \begin{bmatrix} \cos\theta & -\sin\theta & 0 & r\cos\theta \\ \sin\theta & \cos\theta & 0 & r\sin\theta \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We get $q_x = r\cos\theta$

$q_y = r\sin\theta$

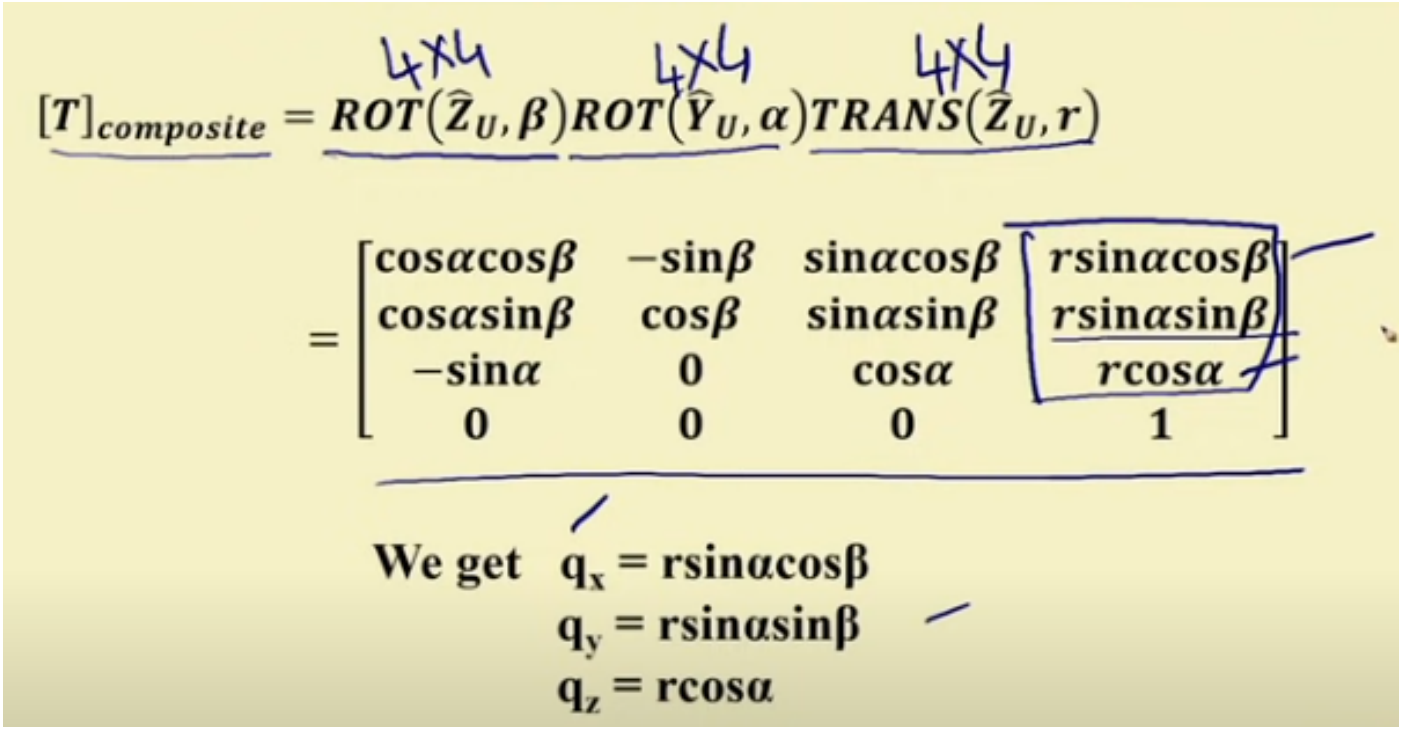
$q_z = z$

Spherical Coordinate System



Steps:

1. Starting from the origin O, translate along $\hat{Z}_U$ axis by r units
2. Rotate in anti-clockwise sense about $\hat{Y}_U$ axis by an angle α
3. Rotate in anti-clockwise sense about $\hat{Z}_U$ axis by an angle β



Detailed discussion of these three coordinate systems:

1. Cartesian Coordinate System:

Imagine a three-dimensional grid with three perpendicular axes: x, y, and z. The point where all axes meet is called the origin (0, 0, 0). Each point in space is uniquely identified by its signed distances (positive or negative) from the origin along each axis. Think of it like a three-dimensional address system.

Strengths:

- Easy to visualize and understand.
- Simple calculations for linear motions.
- Widely used in robotics with linear actuators.

Weaknesses:

- Can be cumbersome for circular or spherical movements.
- Doesn't directly represent angles.

2. Cylindrical Coordinate System:

Picture a cylinder standing upright. You use three parameters to locate a point inside this cylinder:

- **r:** The radial distance from the center of the cylinder to the point.
- **θ:** The angle in the horizontal plane, measured from a reference direction around the cylinder's axis.
- **z:** The height of the point along the cylinder's axis (similar to the z-axis in Cartesian).

Strengths:

- Efficient for robots with rotary and linear joints, like SCARA robots or cylindrical robots.
- Easier calculations for movements along cylindrical paths.

Weaknesses:

- Cannot represent points on the negative z-axis.
- Requires trigonometric conversions for some calculations.

3. Spherical Coordinate System:

Imagine a sphere at the origin. This system uses three parameters to find any point on the sphere:

- **ρ:** The radial distance from the origin to the point.
- **θ:** Same as in the cylindrical system, the angle in the horizontal plane.
- **φ:** The angle measured from the positive z-axis down to the point, ranging from 0° (north pole) to 180° (south pole).

Strengths:

- Efficient for robots with articulated arms, like industrial robots.
- Suitable for analyzing movements around a central point.

Weaknesses:

- Most complex to visualize and requires careful attention to singularities (points at the poles where φ becomes 0° or 180°).
- Requires trigonometric conversions for all calculations

Comparison between Cartesian, Cylindrical and Spherical Coordinate Systems

Feature	Cartesian	Cylindrical	Spherical
Description	Three axes (x, y, z) representing distances	Two distances (r, z) and one angle (θ)	One distance (ρ) and two angles (θ, φ)

Strengths	Simple and intuitive, easy to visualize	Efficient for movements along cylindrical paths, good for robots with rotary and linear joints	Efficient for movements around a central point, good for articulated robots
Weaknesses	Can be cumbersome for circular or spherical movements	Cannot represent points on the negative z-axis	Can be complex to visualize, singularities at poles (φ=0° or 180°)
Common robot examples	Cartesian robots (pick-and-place), gantry robots	SCARA robots, cylindrical robots	Articulated robots (industrial robots, humanoid robots)
Kinematics	Straightforward calculations	Requires trigonometric conversions	Requires trigonometric conversions
Dynamics	Easier to interpret forces and moments	Can be more complex due to angular terms	Most complex, requires careful treatment of singularities

Additional notes:

- **Degrees of freedom:** Cartesian: 3, Cylindrical: 3, Spherical: 3
- **Singularity**: Cartesian: none, Cylindrical: none, Spherical: poles (φ=0° or 180°)
- **Conversion between systems:** Equations exist for conversion between all three systems.

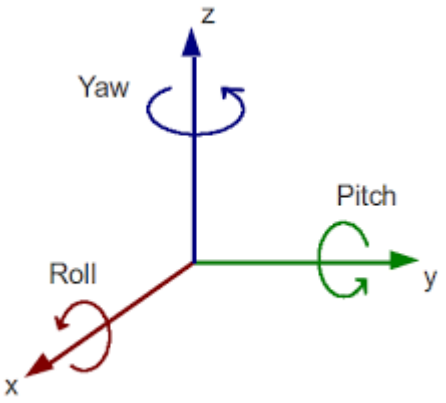
The choice of coordinate system depends on the specific robot and task. Each system has its own advantages and disadvantages, so it is important to consider which one is best suited for the particular application.

Revision

Where,

n= Normal Vector
s= Sliding Vector
a= Approach Vector

Roll Pitch & Yaw Angles (RPY) in Robotics



In robotics, **Roll, Pitch, and Yaw (RPY) angles** are a set of three angles used to describe the **orientation** of a rigid body in three-dimensional space. They offer an intuitive way to represent rotations around the body's own axes, similar to how we describe the head movements of a human or animal.

1. Roll: Imagine the body lying on its side. Roll is the **rotation around the longitudinal axis (X-axis)**, tilting it left or right like rolling a barrel. In an airplane, rolling left makes the wings dip to the left.

2. Pitch: This is the **rotation around the transverse axis (Y-axis)**, tilting the body forward or backward like nodding your head. In an airplane, pitching up points the nose upward.

3. Yaw: Imagine the body standing upright. Yaw is the **rotation around the vertical axis (Z-axis)**, turning it left or right like twisting your torso. In an airplane, yawing left turns the entire aircraft to the left.

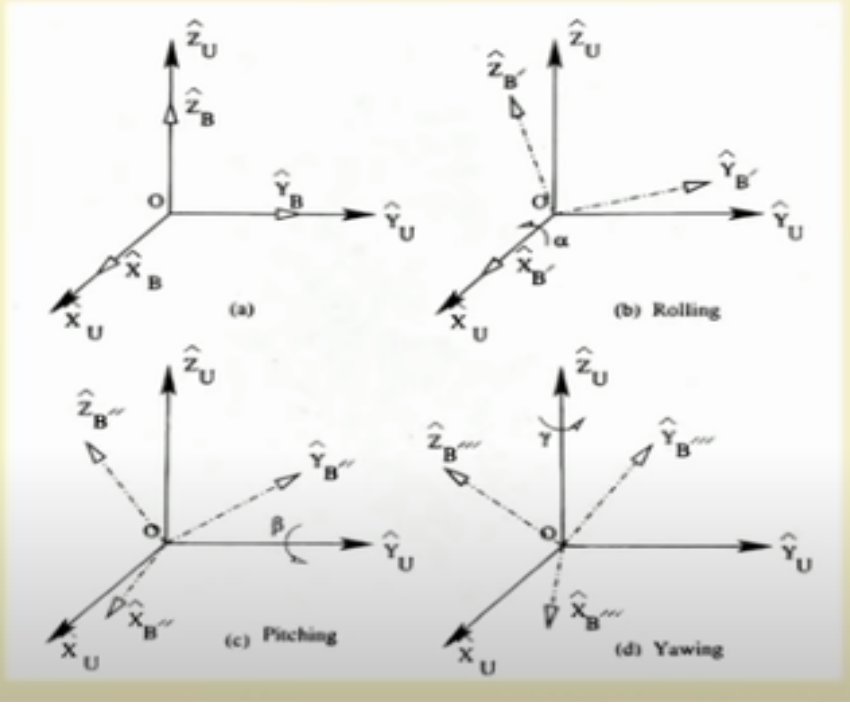
Conventions and Considerations:

- The order of applying rotations (XYZ, XZY, YZX, etc.) can affect the final orientation and needs to be specified.
- RPY angles have limitations, like singularities when pitch reaches 90 degrees, where information is lost.
- Other representations like quaternions can avoid these limitations but may be less intuitive for certain applications.

Applications in Robotics:

- **Controlling robot movements:** RPY angles can be used to program robots to move their manipulators or bodies to specific orientations.
- **Analyzing sensor data:** By interpreting sensor data in the context of the robot's orientation (represented by RPY angles), robots can better understand their environment and make informed decisions.
- **Human-robot interaction:** When robots need to communicate their orientation or respond to human gestures, RPY angles can be helpful for translation.

Roll, Pitch and Yaw Angles



$$ {}^{U}_{B}R_{composite,\,rpy} = ROT(\hat{Z}_U, \gamma)\,ROT(\hat{Y}_U, \beta)\,ROT(\hat{X}_U, \alpha) $$

$$ = \begin{bmatrix} c\beta c\gamma & -c\alpha s\gamma + s\alpha s\beta c\gamma & s\alpha s\gamma + c\alpha s\beta c\gamma \\ c\beta s\gamma & c\alpha c\gamma + s\alpha s\beta s\gamma & -s\alpha c\gamma + c\alpha s\beta s\gamma \\ -s\beta & c\beta s\alpha & c\alpha c\beta \end{bmatrix} $$

We compare with

$$ {}^{U}_{B}R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} $$

$c\beta = \cos\beta$, $s\alpha = \sin\alpha$, $\frac{c\beta\, s\alpha}{c\alpha}$

We get

$$\alpha = \tan^{-1}\left(\frac{r_{32}}{r_{33}}\right)$$

$$\beta = \tan^{-1}\left(\frac{-r_{31}}{\sqrt{r_{11}^2 + r_{21}^2}}\right)$$

$$\gamma = \tan^{-1}\left(\frac{r_{21}}{r_{11}}\right)$$

A Numerical Example

The concept of roll, pitch and yaw angles has been used to represent the rotation of a frame {B} with respect to the reference frame {U}, that is ${}^{U}_{B}R$. Let us suppose that the above rotation can also be expressed by a 3X3 rotation matrix as given below.

$${}^{U}_{B}R = \begin{bmatrix} -0.250 & 0.433 & -0.866 \\ 0.433 & -0.750 & -0.500 \\ -0.866 & -0.500 & 0.000 \end{bmatrix}$$

Determine the angles of rolling, pitching and yawing.

Solution:

$$\text{Angle of rolling } \alpha = \tan^{-1}\frac{r_{32}}{r_{33}} = \tan^{-1}\frac{-0.500}{0.000} = 90°$$

$$\text{Angle of pitching } \beta = \tan^{-1}\frac{-r_{31}}{\sqrt{r_{11}^2 + r_{21}^2}}$$

$$= \tan^{-1}\frac{0.866}{\sqrt{(-0.250)^2 + (0.433)^2}}$$

$$= 40.89°$$

$$\text{Angle of yawing } \gamma = \tan^{-1}\frac{-r_{21}}{r_{11}} = \tan^{-1}\frac{0.433}{-0.250}$$

$$= -59.99 \approx -60°$$

Euler Angles in Robotics

In robotics, **Euler angles**, like Roll, Pitch, and Yaw (RPY), are a type of **representation for the orientation of a rigid body in three-dimensional space**. They describe an object's orientation by specifying three consecutive rotations around designated axes. While RPY uses specific axes based on the body itself (longitudinal, transverse, and vertical), Euler angles offer more flexibility in choosing the rotation axes.

Here's a deeper dive into Euler angles:

Basics:

- **Three angles:** They consist of three individual angles (typically denoted as α, β, γ) applied sequentially about chosen axes.
- **Axis order:** Different conventions define the order of axes for rotation, leading to various combinations like ZYX, XYZ, ZXZ, etc. Each convention has its own

strengths and limitations depending on the robot's structure and the task at hand.
- **Visualizing:** While visualizing RPY is relatively straightforward, understanding Euler angles visually can be trickier due to the varying axis order options.

Strengths:

- **Intuitive:** For certain robot structures and tasks, Euler angles can be an intuitive way to represent orientation, reflecting how humans might think about rotations.
- **Simple calculations:** Depending on the chosen order and axes, calculations using Euler angles can be simpler than other representations like quaternions for specific movements.

Weaknesses:

- **Singularities:** Certain combinations of angles can lead to singularities, where information about the orientation is lost and calculations become undefined. This necessitates careful selection of axes and angle ranges.
- **Redundancy:** Depending on the axis order, multiple sets of Euler angles can represent the same orientation, causing ambiguity.
- **Complexity:** Visualizing and interpreting Euler angles, especially with different conventions, can be more challenging than RPY.

Applications in Robotics:

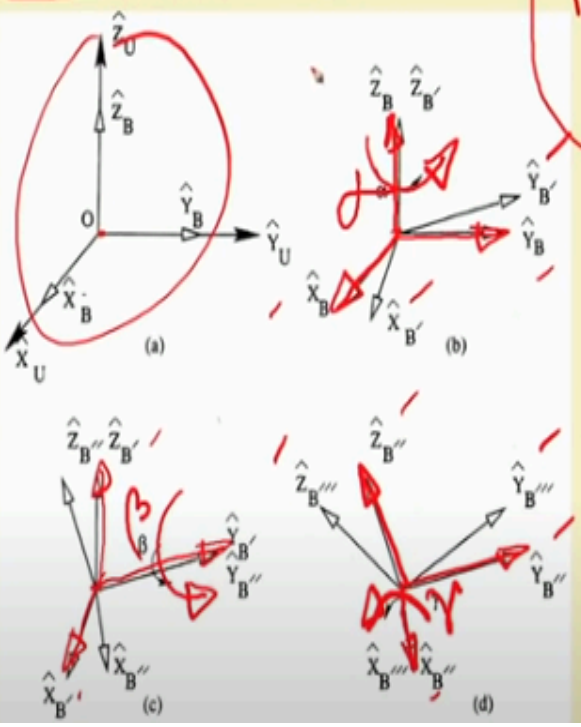
- **Programming robot motions:** With appropriate axis selection and singularity avoidance, Euler angles can be used to program desired robot orientations for tasks like manipulation or navigation.
- **Analyzing sensor data:** Similar to RPY, interpreting sensor data based on the robot's Euler angle orientation can enhance understanding of its environment.
- **Human-robot interaction:** When a robot needs to describe its orientation or respond to user input based on orientation, Euler angles can be a suitable choice depending on the context and complexity.

Choosing the right representation:

While Euler angles offer flexibility and can be intuitive for certain applications, other representations like quaternions can avoid singularities and redundancy, providing a more mathematically robust and globally unique solution for orientation. The choice ultimately depends on the specific robot, task, and desired level of simplicity vs. mathematical rigor

Euler Angles

Using Euler Angles



$${}^{U}_{B}R = {}^{B}_{U}R^{-1}$$

Steps:

1. Rotate {B} about $\hat{Z}_B$ by an angle α in anti-clockwise sense
2. Rotate {B} about $\hat{Y}_{B'}$ by an angle β in anti-clockwise sense
3. Rotate {B} about $\hat{X}_{B''}$ by an angle γ in anti-clockwise sense

$${}^{B}_{U}R_{Eulerangles} = ROT(\hat{X}_{B''}, -\gamma)\,ROT(\hat{Y}_{B'}, -\beta)\,ROT(\hat{Z}_{B}, -\alpha)$$

$${}^{U}_{B}R = \begin{bmatrix} c\alpha c\beta & s\beta s\gamma c\alpha - s\alpha c\gamma & s\beta c\gamma c\alpha + s\alpha s\gamma \\ s\alpha c\beta & s\beta s\gamma s\alpha + c\alpha c\gamma & s\beta c\gamma s\alpha - s\gamma c\alpha \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}$$

We compare with

$${}^{U}_{B}R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Denavit-Hartenberg Approach

Proposed in the year 1955. The Denavit-Hartenberg (DH) approach is a **systematic method for assigning coordinate frames to the links of a serial robot and defining the parameters that describe the transformations between these frames**. This approach is designed specifically for **multi-link** systems. This allows us to calculate the position and orientation of the end-effector based on the joint angles (forward kinematics) and vice versa (inverse kinematics).

Basic principles:

- **Coordinate frames:** The approach assigns a unique coordinate frame to each link and the base of the robot, with each frame having its own X, Y, and Z axes.
- **Four parameters:** Each link is described by four parameters: α, a, d, and θ.
    - **α (alpha):** Angle between the Z-axis of the previous frame and the common perpendicular line between the Z-axes of the current and previous frames.
    - **a (a):** Distance between the Z-axes of the previous and current frames along the common perpendicular line.
    - **d (d):** Offset distance between the joint axis and the common perpendicular line.
    - **θ (theta):** Angle between the X-axes of the current and previous frames about the Z-axis of the previous link.
- **Homogeneous transformation matrices:** These 4x4 matrices represent the transformation between two consecutive frames, incorporating both translation and rotation.

Procedure:

1. Assign a coordinate frame to each link and the base of the robot, ensuring consistent axis orientations.
2. Define the four DH parameters for each link based on the chosen axes and link geometry.
3. Use the DH parameters to construct the homogeneous transformation matrices for each link transformation.
4. Chain the transformation matrices together to calculate the overall transformation from the base frame to the end-effector frame. This gives the end-effector position and orientation for a given set of joint angles (forward kinematics).
5. To solve for joint angles for a desired end-effector pose (inverse kinematics), the process can be reversed, although it may be more complex and require numerical methods.

Advantages & disadvantages of Denavit-Hartenberg approach:

Feature	Advantage	Disadvantage
Systematic and Standardized	- Consistent and well-defined method for modeling robots. - Facilitates comparison and analysis across different systems.	- Not universally applicable to all robot configurations.
Intuitive Parameters	- Clear geometric interpretation of DH parameters, making them easier to understand.	- Sensitivity to axis selection, requiring careful attention to avoid singularities.
Versatility	- Applicable to a wide range of serial robots, regardless of complexity. - Standardized parameters enable sharing and collaboration.	- Potential redundancy with multiple valid parameter sets for the same robot, requiring careful selection.
Numerical Efficiency	- Efficient for numerical calculations, crucial for real-time robot control.	- Less intuitive for highly branched or unusual robot structures, requiring additional considerations.
Ease of Implementation	- Standardized approach simplifies implementation in kinematic models and software.	- Requires careful definition of axes and parameters to avoid errors and singularities.

Additional Considerations:

- Choice of axes can significantly impact parameter values and solutions.
- May not be ideal for robots with closed-loop kinematic chains.
- While efficient, alternative representations like quaternions might offer computational advantages in specific cases.

D-H Parameters (Link & Joint)

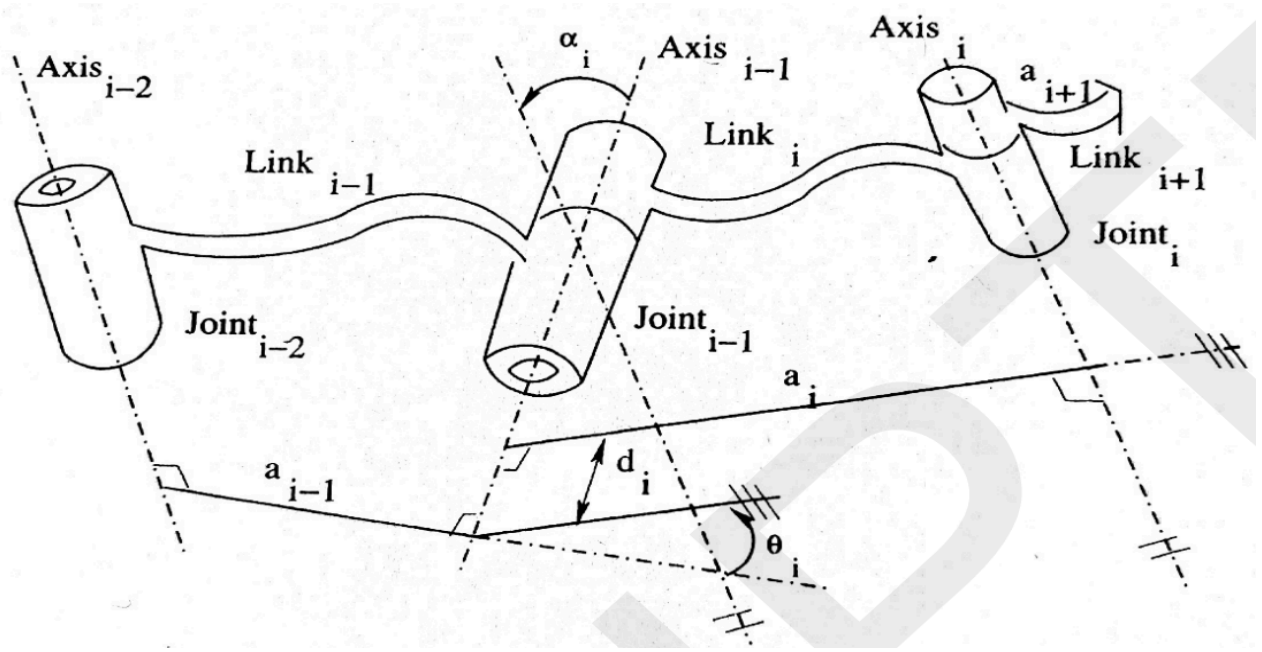
Denavit-Hartenberg (DH) parameters serve as a standardised and systematic approach for analyzing serial robots. They act as a unique **"fingerprint"** that defines the robot's geometry and facilitates calculations related to its end-effector pose.

Key principles:

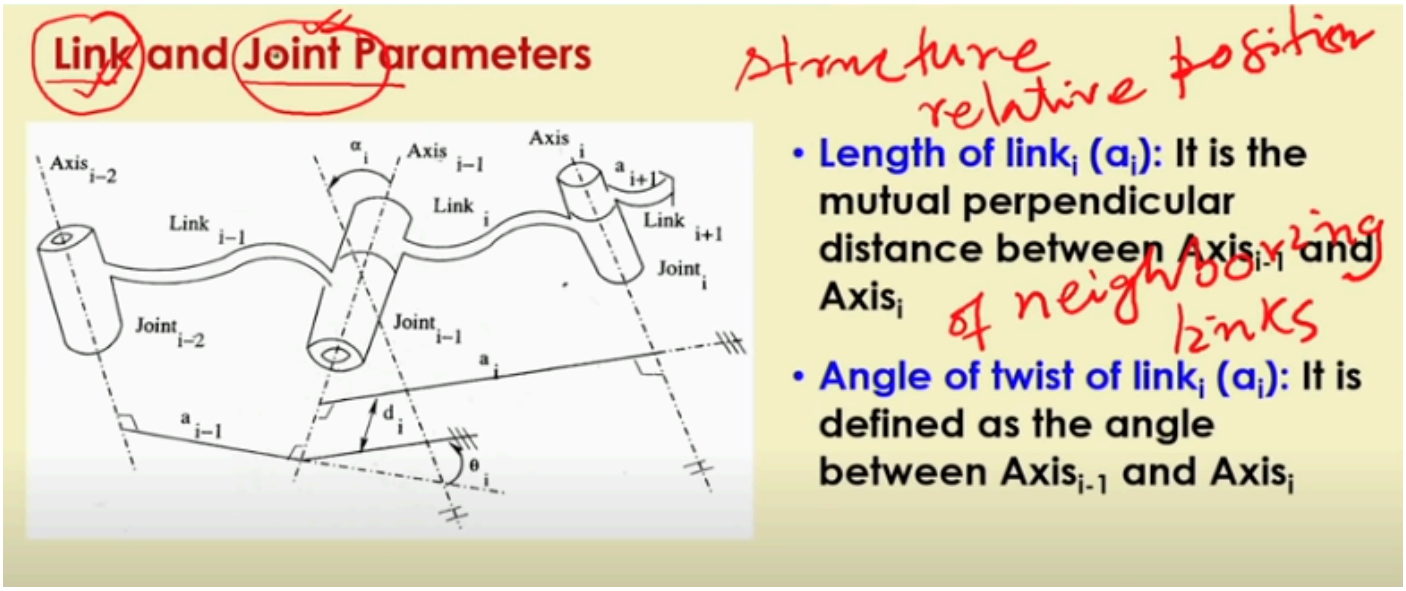
- **Coordinate frames:** Each link and the base of the robot possess a dedicated coordinate frame with orthogonal axes (X, Y, Z).
- **Four parameters:** α, a, d, and θ:
  - α: Angle between successive Z-axes and the common perpendicular line.
  - a: Distance between Z-axes along the common perpendicular line.
  - d: Offset distance between the joint axis and the common perpendicular line.
  - θ: Angle between successive X-axes about the previous Z-axis.
- **Homogeneous transformations:** These 4x4 matrices encapsulate both translation and rotation, representing transformations between frames.

Applications:

1. **Forward Kinematics:** Given joint angles (θs), calculate the end-effector pose (position and orientation). DH parameters define the link transformations, forming the chain that leads to the final pose.
2. **Inverse Kinematics:** Given a desired end-effector pose, determine the necessary joint angles. By working backwards through the transformations defined by DH parameters, we arrive at the required joint values.
3. **Path Planning and Trajectory Generation:** Understanding how joint movements affect the end-effector (thanks to DH parameters) is crucial for planning smooth and accurate robot paths.
4. **Robot Simulation and Analysis:** Accurately modeling robot kinematics for simulation and analysis relies heavily on DH parameters, enabling virtual testing and performance evaluation.



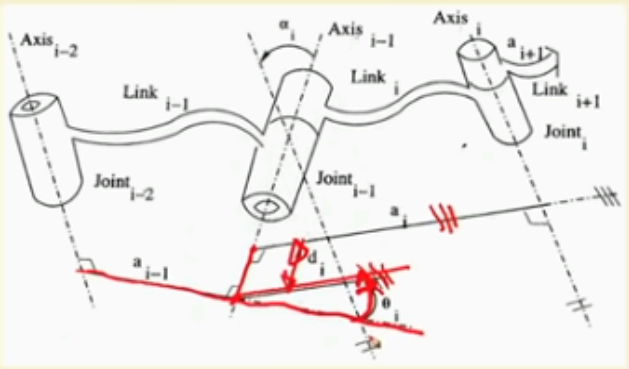
Link Parameters:



Here, remember:

1. **α_i = Angle of twist of link i** *(Correct the printing mistake in above slide)*
2. If **α_i** is rotated in an anticlockwise direction the angle is **$+\alpha_i$.**
3. If **α_i** is rotated in clockwise direction the angle is **$-\alpha_i$.**
4. If both axes are coplanar and intersect at one point then, ***α_i=0.***

Joint Parameters:



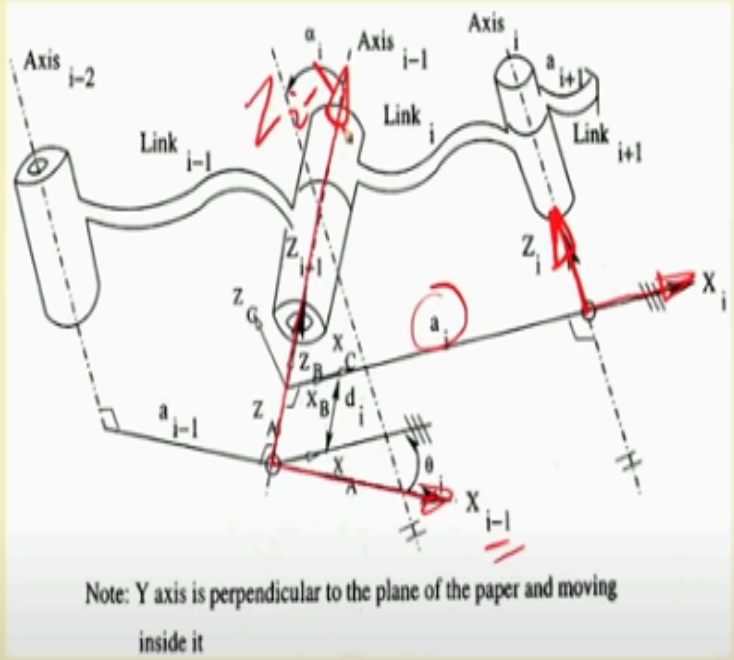
- **Offset of link$_i$ (d_i):** It is the distance measured from a point where a_{i-1} intersects the Axis$_{i-1}$ to the point where a_i intersects the Axis$_{i-1}$ measured along the said axis
- **Joint Angle (θ_i):** It is defined as the angle between the extension of a_{i-1} and a_i measured about the Axis$_{i-1}$

Notes:
- Revolute joint: θ_i is variable
- Prismatic joint: d_i is variable

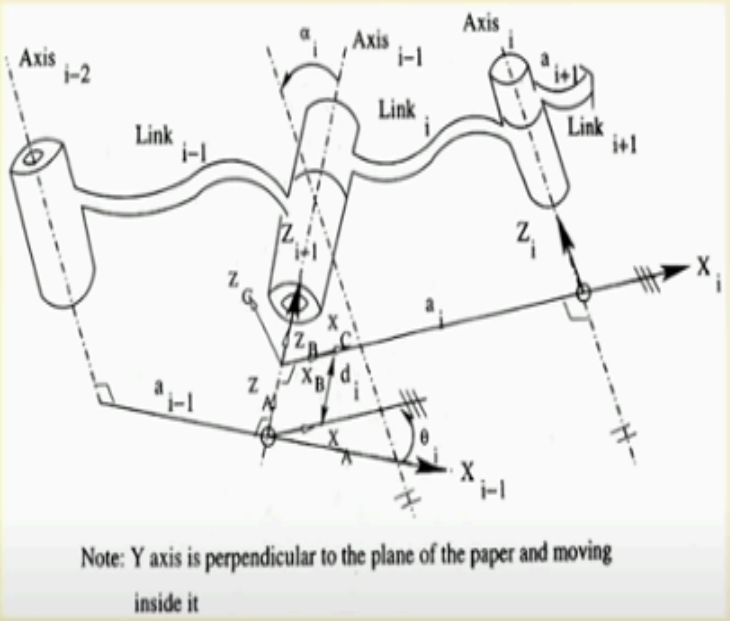
Here, remember:

1. For Revolute Joints: **θ_i is variable and α_i, a_i and d_i are constant.**
2. For Prismatic Joint: **d_i is variable and α_i, a_i and θ_i are constant.**

Rules for Coordinate Assignment



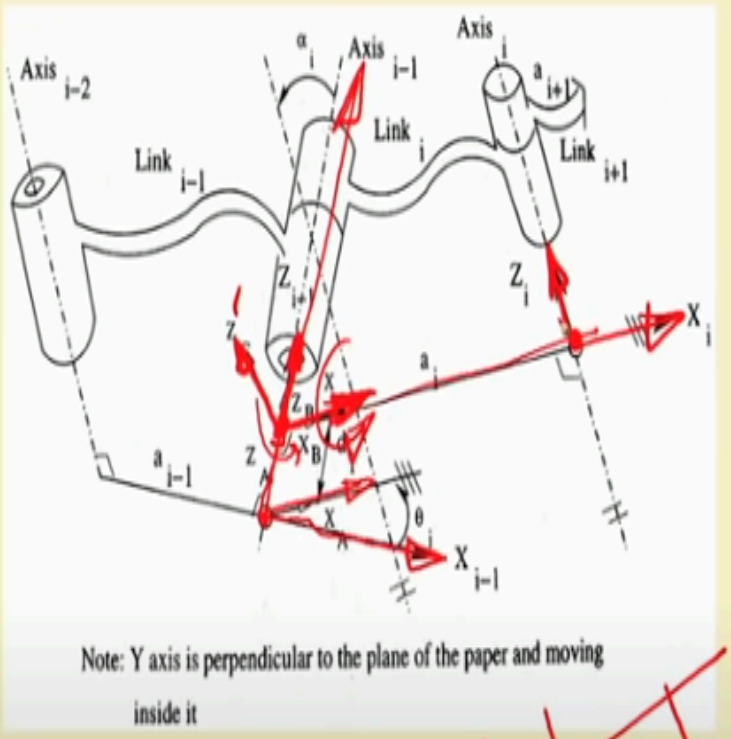
- Z_i is an axis about which the rotation is considered or along which the translation takes place
- If Z_{i-1} and Z_i axes are parallel to each other, X axis will be directed from Z_{i-1} to Z_i along their common normal



- If Z_{i-1} and Z_i axes intersect each other, X axis can be selected along either of two remaining directions
- If Z_{i-1} and Z_i axes act along a straight line, X axis can be selected anywhere in a plane perpendicular to them
- Y axis is decided as Y = ZxX

We have

$$
\begin{aligned}
{}^{i-1}_{i}T &= {}^{i-1}_{A}T\,{}^{A}_{B}T\,{}^{B}_{C}T\,{}^{C}_{i}T \\
&= ROT(Z,\theta_i)\,TRANS(Z,d_i)\,ROT(X,\alpha_i)\,TRANS(X,a_i) \\
&= Screw_Z\,Screw_X
\end{aligned}
$$



- If Z_{i-1} and Z_i axes intersect each other, X axis can be selected along either of two remaining directions
- If Z_{i-1} and Z_i axes act along a straight line, X axis can be selected anywhere in a plane perpendicular to them
- Y axis is decided as Y = ZxX

$$ {}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & -s\theta_i c\alpha_i & s\theta_i s\alpha_i & a_i c\theta_i \\ s\theta_i & c\theta_i c\alpha_i & -c\theta_i s\alpha_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Now, $ {}^{i}_{i-1}T = \left[{}^{i-1}_{i}T \right]^{-1} $

$$ = \begin{bmatrix} c\theta_i & s\theta_i & 0 & -a_i \\ -s\theta_i c\alpha_i & c\theta_i c\alpha_i & s\alpha_i & -d_i s\alpha_i \\ s\theta_i s\alpha_i & -c\theta_i s\alpha_i & c\alpha_i & -d_i c\alpha_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Strengths:

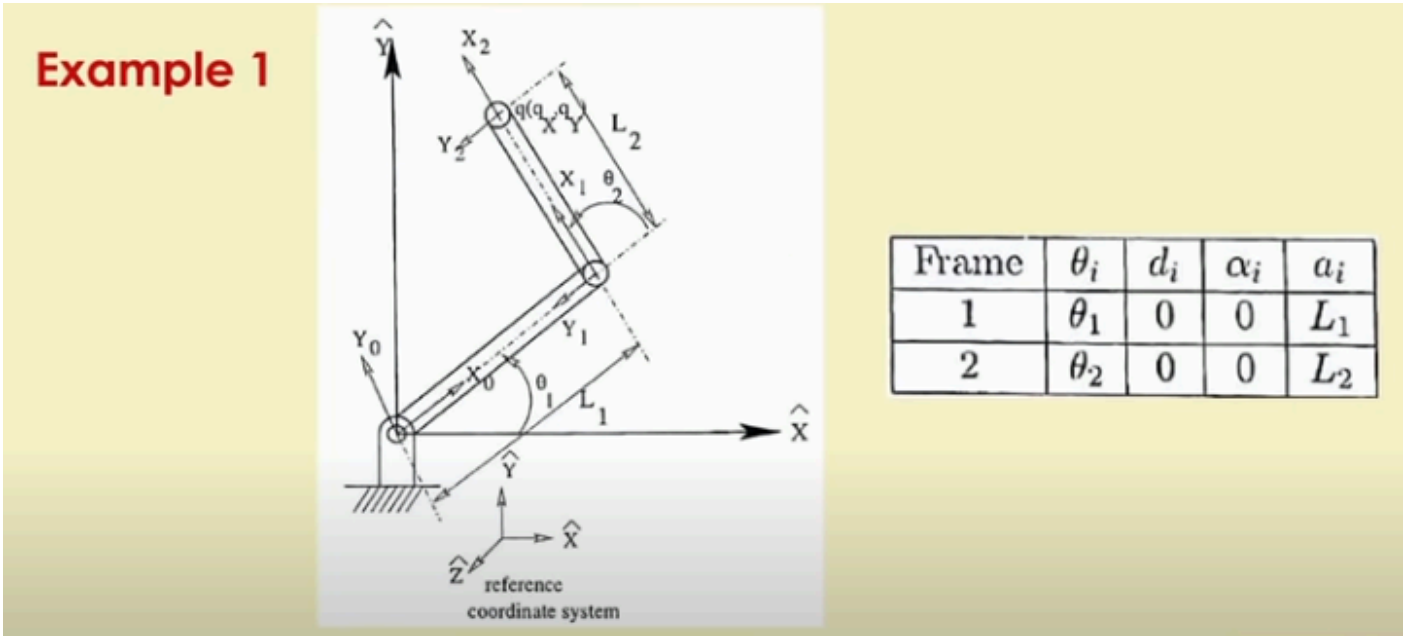
- Intuitive interpretation of robot geometry.
- Efficient for numerical calculations in real-time control.
- Standardised for easier understanding and implementation.

Limitations:

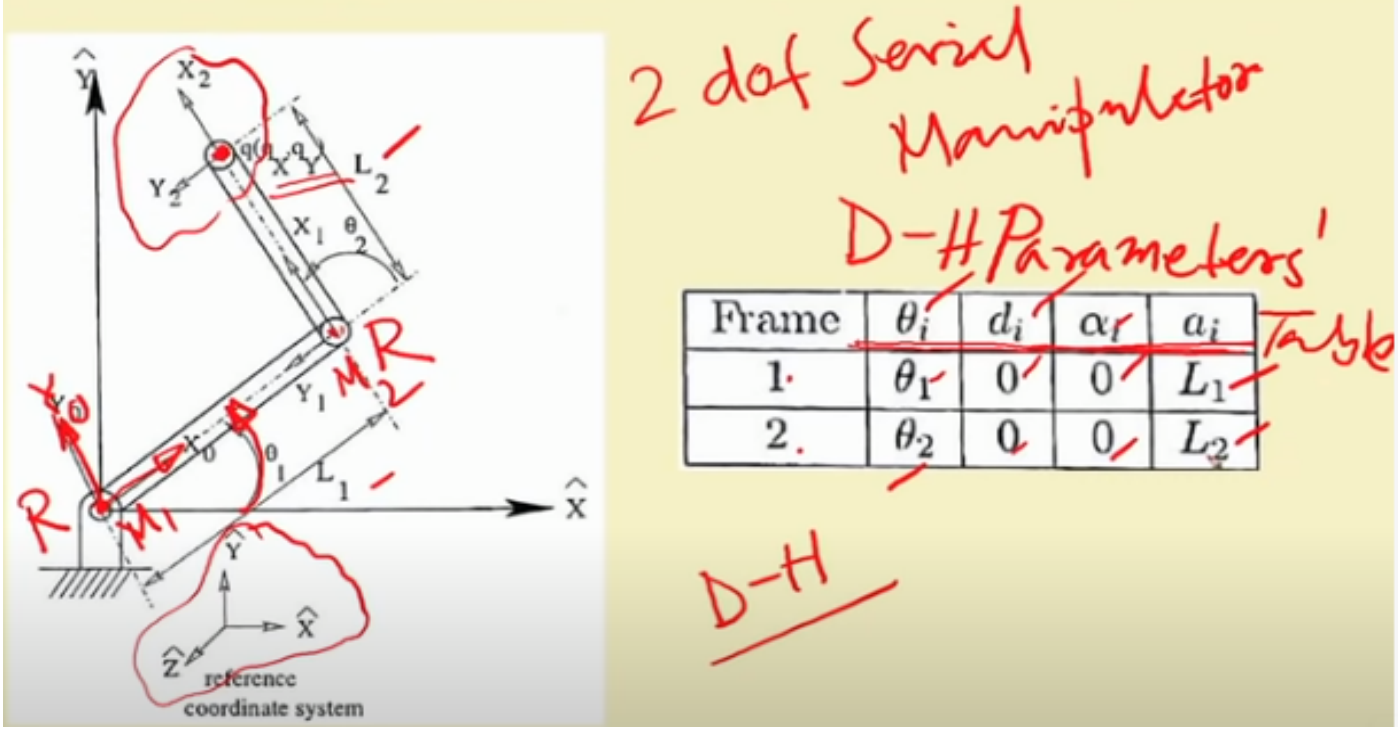
- Sensitive to axis selection, requiring careful attention to avoid singularities.
- Potential redundancy with multiple valid parameter sets for the same robot.
- Less intuitive for highly branched or unusual robot structures.

DH parameters offer a powerful tool for analyzing and controlling serial robots, especially in kinematic applications. Their standardized nature, intuitive interpretation, and efficiency make them essential for various robotic tasks, solidifying their role as the "fingerprint" for understanding and controlling robot motion.

Numerical: Problem of 2 DOF serial Manipulator solving using D-H Parameters



Frame	θ_i	d_i	α_i	a_i
1	θ_1	0	0	L_1
2	θ_2	0	0	L_2



Here,

- Joint 1 - Rotary joint, Joint 2 - Revolute Joint
- M1 & M2 are Motors at Joints 1 and 2
- Assign coordinate systems using DH parameters to all joints
- Z axis will be perpendicular to the paper, so X axis and Y axis are allotted
- Prepare D-H Parameter Table where Sequence is important
- Follow Screw$_X$ Srew$_Z$ rule.

Forward Kinematics

$$ {}^{Base}_{2}T = {}^{Base}_{1}T\,{}^{1}_{2}T $$

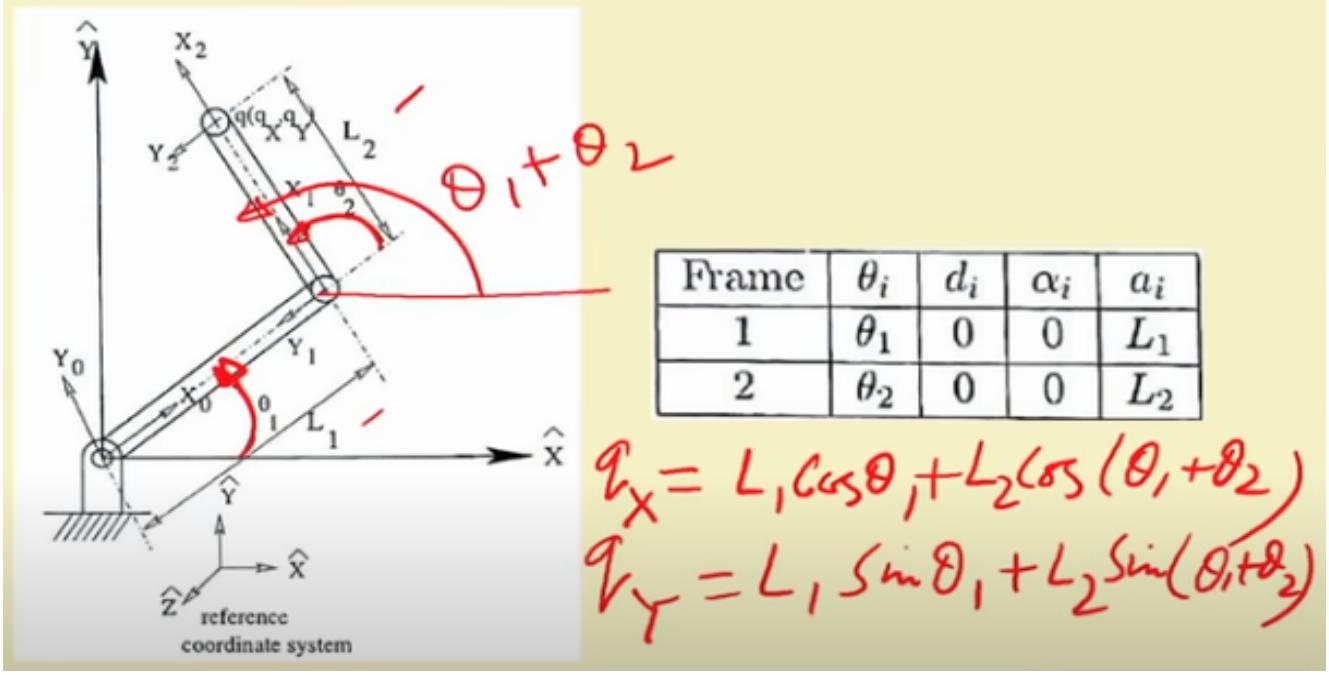
$$ {}^{Base}_{1}T = ROT(\hat{Z},\theta_1)TRANS(\hat{X},L_1) $$

$$ = \begin{bmatrix} c_1 & -s_1 & 0 & L_1c_1 \\ s_1 & c_1 & 0 & L_1s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$
{}^{1}_{2}T = ROT(\hat{Z},\theta_2)TRANS(\hat{X},L_2)
$$

$$
= \begin{bmatrix} c_2 & -s_2 & 0 & L_2c_2 \\ s_2 & c_2 & 0 & L_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

We know that for q_x and q_y.

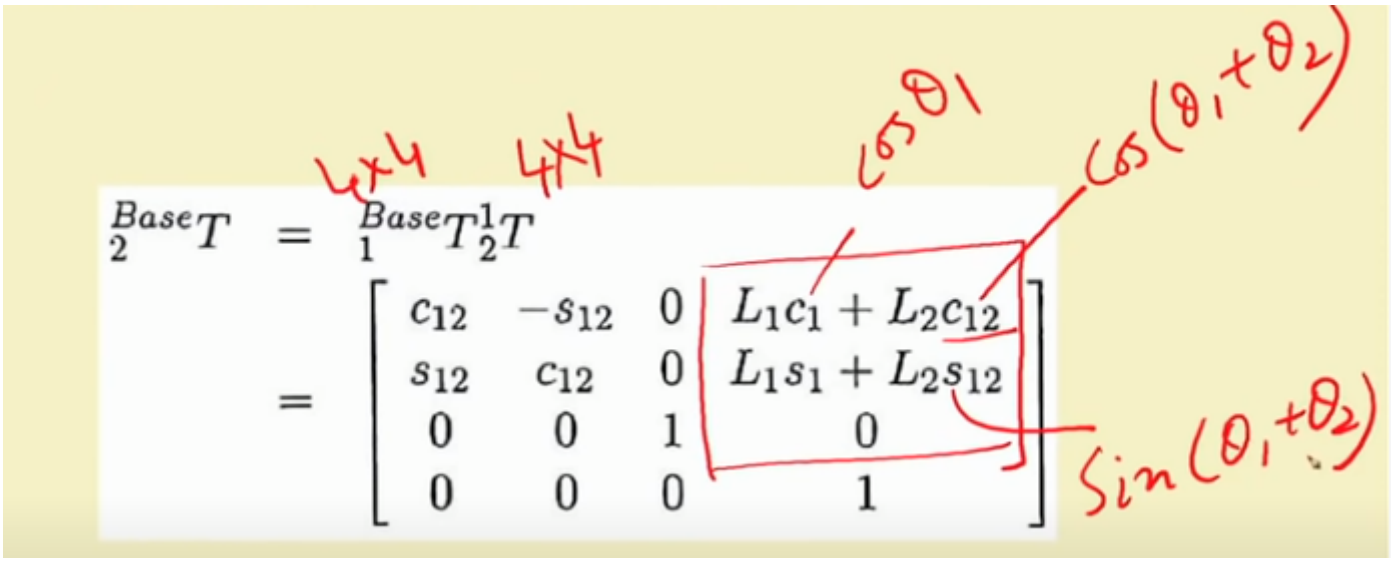


Frame	θ_i	d_i	α_i	a_i
1	θ_1	0	0	L_1
2	θ_2	0	0	L_2

Thus, considering q_x and q_y we get,

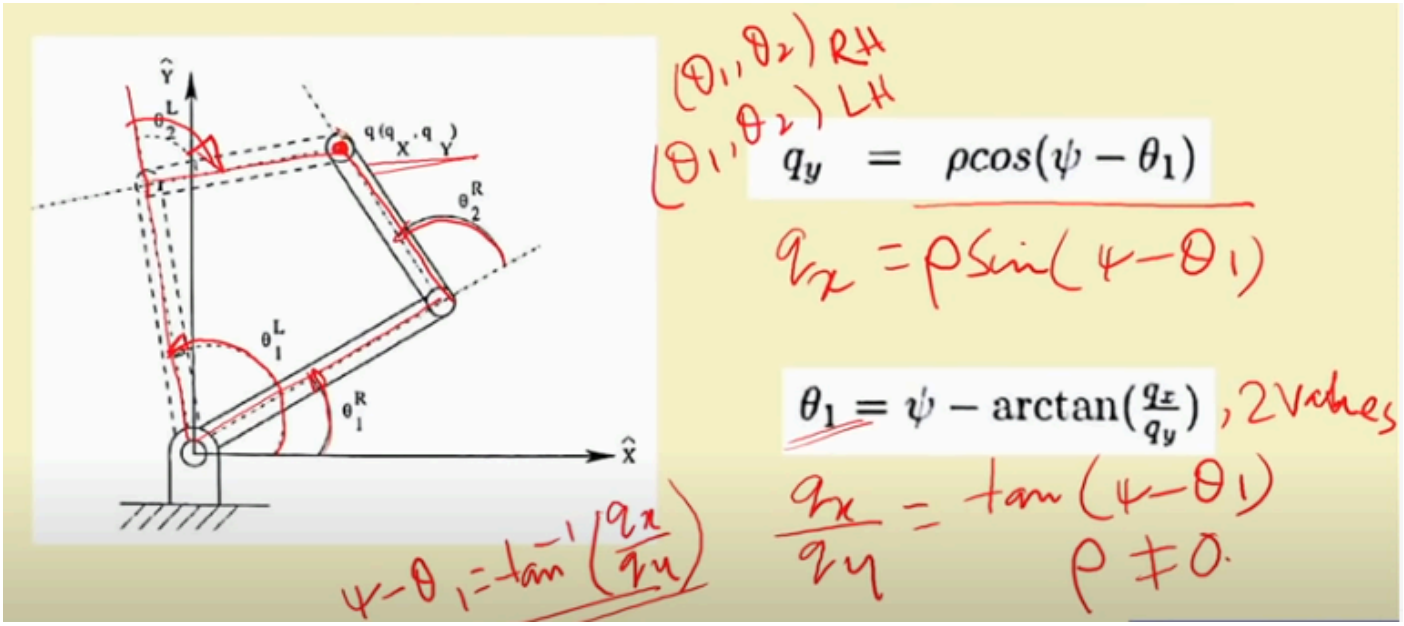
$$
{}^{Base}_{2}T = {}^{Base}_{1}T\,{}^{1}_{2}T
$$

$$
= \begin{bmatrix} c_{12} & -s_{12} & 0 & L_1c_1 + L_2c_{12} \\ s_{12} & c_{12} & 0 & L_1s_1 + L_2s_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} = q_x \\ = q_Y \\ \\ \\ \end{matrix}
$$



$$
{}^{Base}_{2}T = {}^{Base}_{1}T\,{}^{1}_{2}T
$$

$$
= \begin{bmatrix} c_{12} & -s_{12} & 0 & L_1c_1 + L_2c_{12} \\ s_{12} & c_{12} & 0 & L_1s_1 + L_2s_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$



Forward and Inverse Kinematics in Robotics

Introduction:

In robotics, controlling the movement of a robotic arm is crucial for performing various tasks. This lecture explores two fundamental concepts: **forward kinematics** and **inverse kinematics**, which govern the relationship between the joint angles of the arm and the position and orientation of its end effector (the tool or gripper at the tip).

Forward Kinematics:

- **Definition:** Forward kinematics is the process of **calculating the position and orientation of the end effector** of a robotic arm **given the known values of its joint angles**.
- **Analogy:** Imagine a puppet. You know how much you pull each string (joint angles), and forward kinematics tells you **where the puppet's hand will be** (position and orientation).
- **Applications:**
    a. Predicting the robot's movement based on specified joint angles.
    b. Visualizing the robot's path during simulations.

Inverse Kinematics:

- **Definition:** Inverse kinematics is the **opposite of forward kinematics**. It involves **calculating the necessary joint angles** required to **achieve a desired position and orientation of the end effector**.
- **Analogy:** Still using the puppet, you now want its hand to touch a specific point (desired pose). Inverse kinematics helps you figure out **how much to pull each string** (joint angles) to make that happen.
- **Applications:**
    a. Programming robots to reach specific locations and grasp objects.
    b. Motion planning and trajectory control.

Key Points:

- Both forward and inverse kinematics are crucial for **controlling and programming robots**.
- **Forward kinematics** helps **predict** movements, while **inverse kinematics** enables **achieving specific tasks**.
- Solving **inverse kinematics can be more complex**, especially for robots with many degrees of freedom (DOF).

Real-world Examples:

- **Industrial robots:** Use forward kinematics to track the end effector's position during welding or painting tasks. Inverse kinematics helps them reach specific points on objects for assembly or manipulation.
- **Prosthetic limbs:** Forward kinematics helps predict the movement of the prosthetic arm based on muscle signals. Inverse kinematics calculates the motor commands needed to achieve desired hand positions.

Unit 3: Robot Dynamics

Prerequisites for Robot Dynamics:

1. *Robot Kinematics*
2. *Trajectory Planning*

Introduction to Robot Dynamics

Robot dynamics deals with the relationship between the forces acting on a robot and the resulting motion (acceleration and trajectory) of the robot. It's essentially understanding how **forces translate into movement**.

- **Forces and Torques:** These external forces (like gravity, friction, or actuator forces) and torques (twisting forces) act on the robot and influence its motion.
- **Robot Kinematics:** This field provides the mathematical relationships between joint angles (positions and velocities) and the resulting end-effector position and velocity (ignoring forces).
- **Robot Dynamics:** Builds upon kinematics and considers the forces and torques to calculate the robot's acceleration and predict its movement.

Two Main Problems in Robot Dynamics:

1. **Forward Dynamics:** Given the forces and torques acting on the robot and its current configuration (joint angles and velocities), this problem calculates the resulting accelerations of the robot's joints. This is useful for simulations and predicting robot behaviour under different loads.
2. **Inverse Dynamics:** Given the desired trajectory (position and velocity) of the end-effector, this problem calculates the necessary joint torques needed to achieve that desired motion. This is crucial for robot control algorithms to generate the right commands for the motors.

Benefits of Understanding Robot Dynamics:

- **Improved Control:** Enables precise control of robot movements by considering the effects of forces and torques.
- **Trajectory Optimization:** Helps optimize robot movements for efficiency and speed by accounting for dynamics.
- **Simulation and Analysis:** Allows simulation of robot behavior under various conditions before real-world deployment.

- **Design Optimization:** Provides insights for designing robots with better performance and efficiency based on dynamic considerations.

Overall, robot dynamics is a fundamental aspect of robotics. It bridges the gap between robot design and control by providing the crucial link between forces and resulting movements. This knowledge is essential for engineers and researchers working on designing, controlling, and optimizing robotic systems.

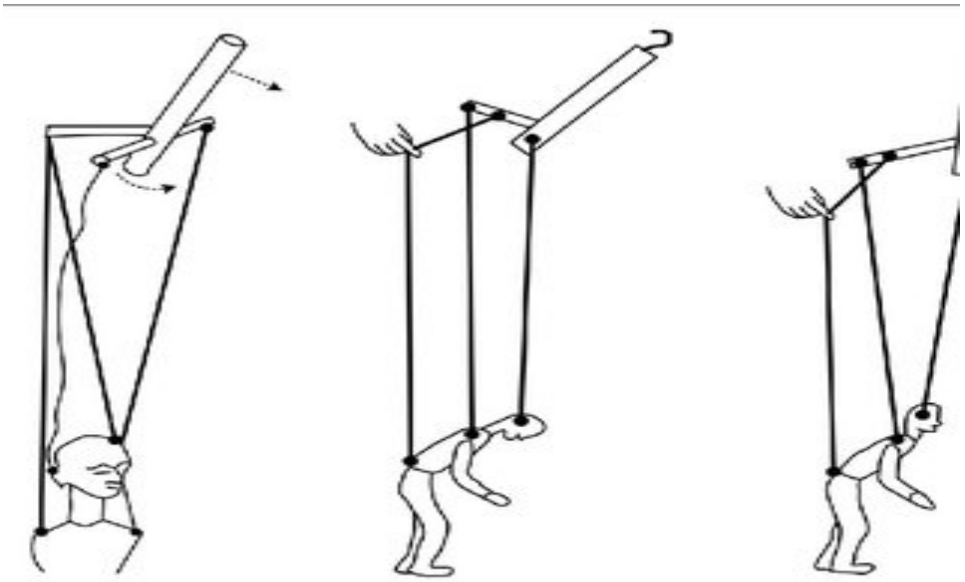
Comparison between Robot Kinematics & Robot Dynamics

Feature	Robot Kinematics	Robot Dynamics
Focus	Geometric relationships between robot joints and end-effector	Relationship between forces/torques and robot motion (acceleration and trajectory)
Inputs	Joint angles (positions and velocities)	Forces and torques acting on the robot, current configuration (joint angles and velocities)
Outputs	End-effector position and velocity (ignoring forces)	Joint accelerations (forward dynamics) or joint torques needed for desired trajectory (inverse dynamics)
Applications	* Robot path planning * Trajectory generation	* Robot control algorithm design * Simulation and analysis of robot behavior * Optimization of robot movements
Complexity	Relatively simpler, focuses on geometric relationships	More complex, involves solving equations of motion
Ignores	Forces and torques acting on the robot	Kinematic relationships (assumes perfect joints with no friction)

Additional Notes:

- Robot dynamics builds upon robot kinematics. Kinematic relationships provide the foundation for understanding how joint angles translate into end-effector motion.
- Dynamics adds the crucial element of forces and torques to predict how the robot will realistically move and interact with its environment.

- Both kinematics and dynamics are essential for robot control. Kinematics helps plan the robot's path, while dynamics helps calculate the necessary forces or torques from the actuators to achieve that desired motion.



Analogy:

- Imagine a puppet. Kinematics is like knowing how pulling various strings will change the hand's position in space (ignoring gravity). Dynamics is like taking gravity into account and calculating how much to pull each string to make the hand reach a specific point while considering the weight of the puppet's arm.

Trajectory Planning

In robotics, **trajectory** refers to the **planned path** a robot's end-effector (gripper or tool) needs to follow to complete a specific task. It essentially defines the **position and orientation** of the end-effector as a function of time.

- **Planning vs. Execution:** Trajectory planning involves determining the desired path for the robot to achieve its goal. Trajectory execution involves controlling the robot's movements to follow the planned path accurately.
- **Components:** A trajectory can be represented by specifying the:
    - **End-effector's position (X, Y, Z coordinates) as a function of time (t).**
    - **Orientation of the end-effector:** This can be represented using various methods like Euler angles, quaternions, or rotation matrices.
    - **Time profile:** This specifies how fast or slow the robot needs to move along the path at different points.

Importance of Trajectory Planning:

- **Accuracy and Efficiency:** Ensures the robot reaches its target location precisely while optimizing movement time and energy consumption.
- **Collision Avoidance:** Plans a path that avoids obstacles and ensures the safety of the robot and its surroundings.
- **Smooth Motion:** Generates a smooth and continuous path to minimize jerks and vibrations, improving the robot's performance and reducing wear and tear.

Types of Trajectories:

- **Point-to-point (PTP):** The robot moves from one point to another in a straight line (linear interpolation) or along a predefined path (e.g., circular).
- **Continuous-path (CP):** The robot needs to follow a continuous path while maintaining a specific orientation (e.g., painting a curved surface).

Trajectory planning algorithms consider various factors like:

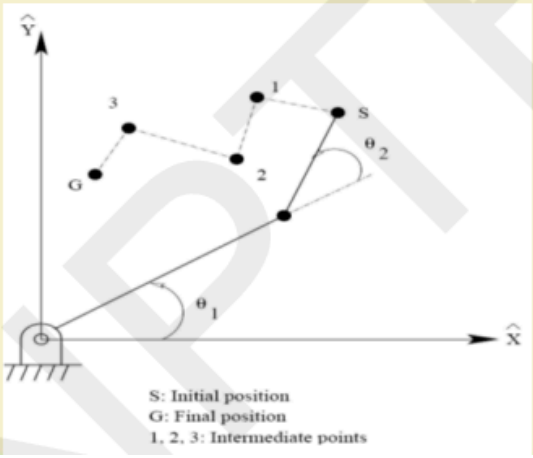
- **Robot Kinematics:** The physical limitations of the robot's joints and their movement capabilities.
- **Dynamics:** The forces and torques acting on the robot that influence its motion.
- **Obstacles:** The presence of obstacles in the environment that the robot needs to avoid.
- **Task Constraints:** Specific requirements like maintaining a certain tool orientation or following a specific velocity profile.

Overall, trajectory planning is a crucial aspect of robot control. By carefully planning the robot's path and movement, engineers can ensure efficient, accurate, and safe operation of robots in various applications.

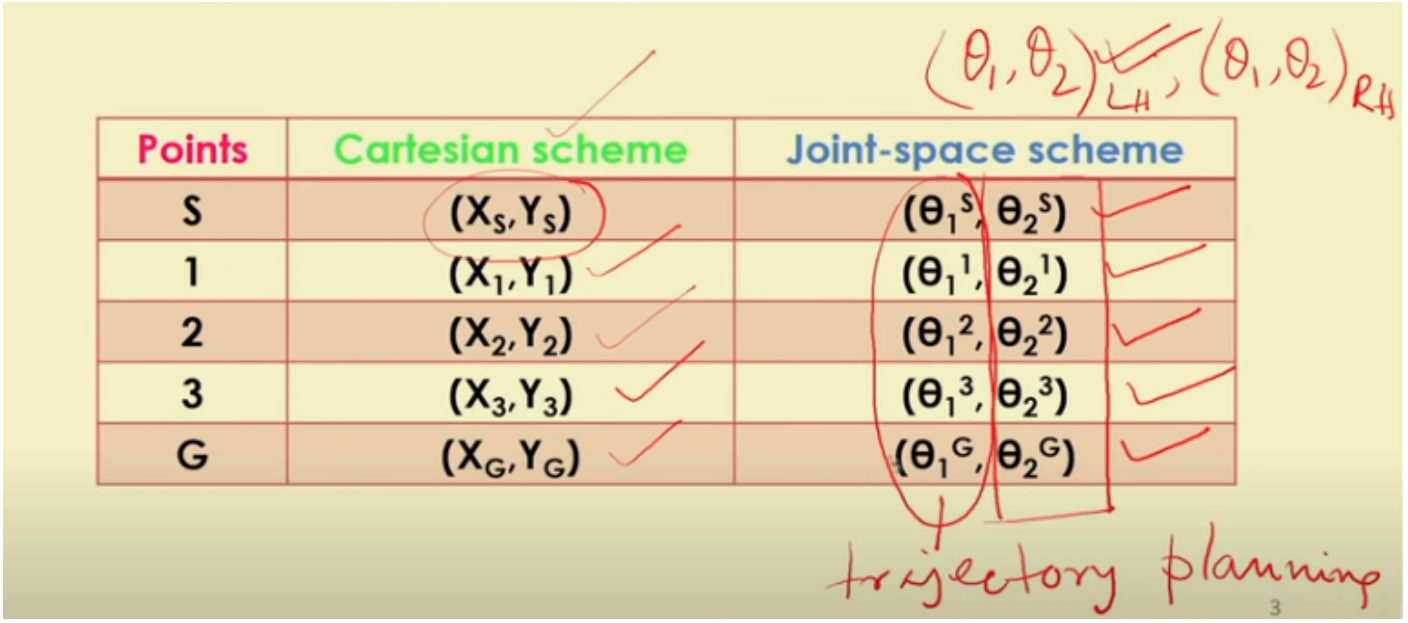
Notes:

- *Rotary Joint: Find out Torque*
- *Linear Joint: Find out Force*
- *Ensure that the Joint angle must vary very smoothly with time so first order derivative (Angular velocity) and second order derivative (Angular acceleration) have to be continuous.*

Aim: To determine time history of position, velocity and acceleration of end-effector of a manipulator, while moving from an initial position to a final position through some intermediate/via points.

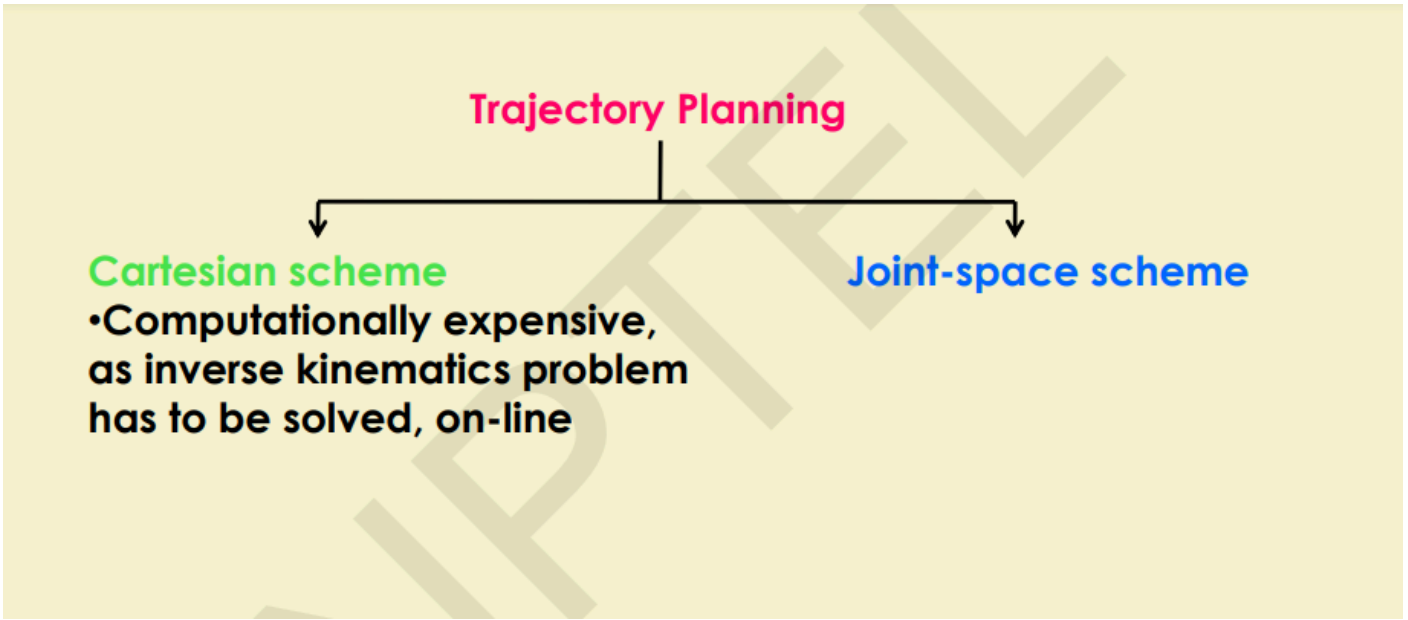


Points	Cartesian scheme	Joint-space scheme
S	(X_S, Y_S)	(θ_1^S, θ_2^S)
1	(X_1, Y_1)	(θ_1^1, θ_2^1)
2	(X_2, Y_2)	(θ_1^2, θ_2^2)
3	(X_3, Y_3)	(θ_1^3, θ_2^3)
G	(X_G, Y_G)	(θ_1^G, θ_2^G)



Points	Cartesian scheme	Joint-space scheme
S	(X_S, Y_S)	(θ_1^S, θ_2^S)
1	(X_1, Y_1)	(θ_1^1, θ_2^1)
2	(X_2, Y_2)	(θ_1^2, θ_2^2)
3	(X_3, Y_3)	(θ_1^3, θ_2^3)
G	(X_G, Y_G)	(θ_1^G, θ_2^G)

We will try to fit a smooth curve from



Joint-Space Scheme (Using JSS to meet the Aim defined above)

$\dot{\theta}$ **= Angular Velocity**
θ(double dot) =Angular acceleration

Joint-Space Scheme

- To fit a smooth (continuous) curve through (θ_1^S, θ_1^1, θ_1^2, θ_1^3, θ_1^G)
- First and second order derivatives must be continuous.

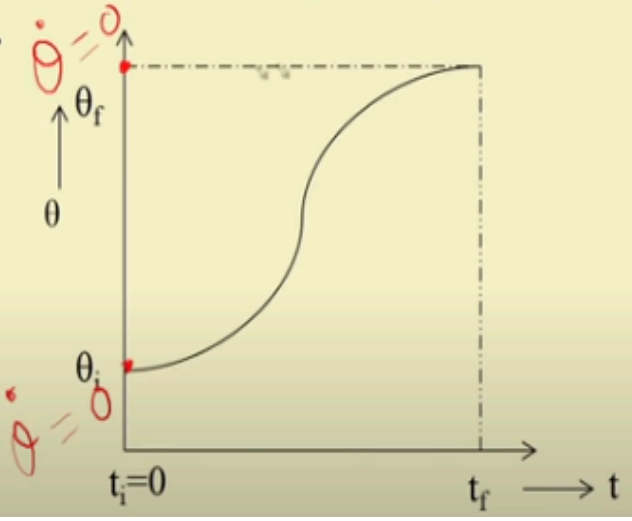
Various Trajectory Functions

- Cubic polynomial
- Fifth-order polynomial
- Linear trajectory function

POLYNOMIAL TRAJECTORY FUNCTION

Case-1

Initial and final values of joint angle are known, and angular velocities at the beginning and end of the cycle are kept equal to zero.



At $t = t_i = 0$; $\theta = \theta_i$, $\dot{\theta} = 0$ $\dot{\theta} = 0$

At $t = t_f$; $\theta = \theta_f$, $\dot{\theta} = 0$ $\dot{\theta} = 0$

$\theta(t)$

- So, as above, we get 4 known conditions.

(The printing error with theta dot has been corrected above)

Let us consider cubic polynomial

$$\theta(t) = C_0 + C_1 t + C_2 t^2 + C_3 t^3$$

where C_0, C_1, C_2, C_3 are the coefficients.

Differentiate $\theta(t)$ with respect to time to get angular velocity

$$\dot{\theta}(t) = C_1 + 2C_2 t + 3C_3 t^2$$

- So, as above, we get 2 things, **Angular displacement** and **Angular velocity**
- Putting the 4 known conditions in these above two equations of **Angular displacement** and **Angular velocity.**

Apply the initial conditions to angular displacement and velocity equations. We get,

$$C_0 = \theta_i \quad (1)$$

$$C_1 = 0 \quad (2)$$

$$C_0 + C_1 t_f + C_2 t_f^2 + C_3 t_f^3 = \theta_f \quad (3)$$

$$C_1 + 2C_2 t_f + 3C_3 t_f^2 = 0 \quad (4)$$

- Putting the 4 known conditions in these above two equations of **Angular displacement** and **Angular velocity,** we get the above 4 equations.
- Here, values of C_0 and C_1 are already known in eq. 1 and eq. 2. Substituting them in eq. 3 & eq. 4.

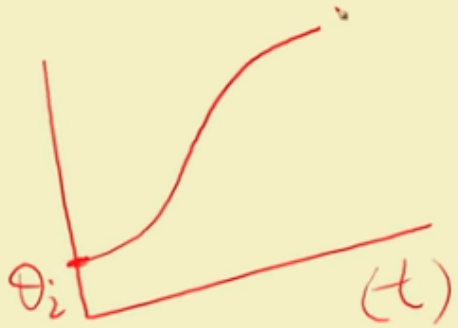
Apply the initial conditions to angular displacement and velocity equations. We get,

$$C_0 = \theta_i$$

$$C_1 = 0$$

$$C_0 + C_1 t_f + C_2 t_f^2 + C_3 t_f^3 = \theta_f$$

$$C_1 + 2C_2 t_f + 3C_3 t_f^2 = 0$$



Solving above equations, We get

$$\theta(t) = \theta_i + \frac{3(\theta_f - \theta_i)}{t_f^2} t^2 - \frac{2(\theta_f - \theta_i)}{t_f^3} t^3$$

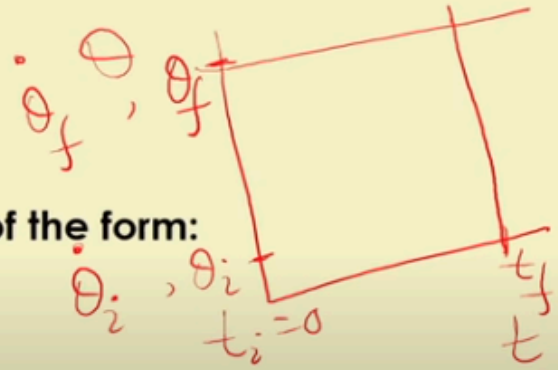
- **θ**(t) is a function of time and in the solution above we can see a cubic polynomial which can create a smooth curve shown above when plotted.
- **Therefore**, with a cubic polynomial we can obtain a smooth curve in the said manner.
- Now, let's explore case 2.

Case-2

Initial and final values of joint angle are known and angular velocities at the beginning and end of the cycle are assumed to have non zero values.

At $t = t_i = 0;\ \theta = \theta_i,\ \dot{\theta} = \dot{\theta}_i$

At $t = t_f;\ \theta = \theta_f,\ \dot{\theta} = \dot{\theta}_f$



Let us consider a third order polynomial of the form:

- Let us try to fit **a cubic polynomial** to the curve using the **four known conditions** shown above.

Let us consider a third order polynomial of the form:

$$\theta(t) = C_0 + C_1 t + C_2 t^2 + C_3 t^3$$

where C_0, C_1, C_2, C_3 are the coefficients.

Differentiate $\theta(t)$ with respect to time to get angular velocity

$$\dot{\theta}(t) = C_1 + 2C_2 t + 3C_3 t^2$$

Apply the initial conditions to angular displacement and velocity equations. We get,

$$C_0 = \theta_i \quad (1)$$

$$C_1 = \dot{\theta}_i \quad (2)$$

$$C_0 + C_1 t_f + C_2 t_f^2 + C_3 t_f^3 = \theta_f \quad (3)$$

$$C_1 + 2C_2 t_f + 3C_3 t_f^2 = \dot{\theta}_f \quad (4)$$

- So here, we have obtained 4 equations with 4 unknowns, solve the equations to obtain the four variables.

Solving above equations, We get

$$C_0 = \theta_i,$$

$$C_1 = \dot{\theta}_i$$

$$C_2 = \frac{3(\theta_f - \theta_i)}{t_f^2} - \frac{2}{t_f}\dot{\theta}_i - \frac{1}{t_f}\dot{\theta}_f$$

$$C_3 = -\frac{2(\theta_f - \theta_i)}{t_f^3} + \frac{1}{t_f^2}(\dot{\theta}_f + \dot{\theta}_i).$$

Case-3

Initial and final values of joint angle are known and angular velocities and accelerations at the beginning and end of the cycle are assumed to have non zero values.

At $t = t_i = 0;\ \theta = \theta_i,\ \dot{\theta} = \dot{\theta}_i,\ \ddot{\theta} = \ddot{\theta}_i$

At $t = t_f;\ \theta = \theta_f,\ \dot{\theta} = \dot{\theta}_f,\ \ddot{\theta} = \ddot{\theta}_f$

Let us consider a fifth-order polynomial as follows:

$$\theta(t) = C_0 + C_1 t + C_2 t^2 + C_3 t^3 + C_4 t^4 + C_5 t^5$$

where C_0, C_1, C_2, C_3, C_4 **and** C_5 **are the coefficients.**

Differentiate $\theta(t)$ with respect to time once to get angular velocity and twice to get angular acceleration

$$\dot{\theta}(t) = c_1 + 2c_2t + 3c_3t^2 + 4c_4t^3 + 5c_5t^4$$

$$\ddot{\theta}(t) = 2c_2 + 6c_3t + 12c_4t^2 + 20c_5t^3$$

Apply the initial conditions to angular displacement, velocity and acceleration equations. We get,

$$c_0 = \theta_i \quad (1)$$

$$c_1 = \dot{\theta}_i \quad (2)$$

$$c_2 = \frac{1}{2}\ddot{\theta}_i \quad (3)$$

$$c_0 + c_1t_f + c_2t_f^2 + c_3t_f^3 + c_4t_f^4 + c_5t_f^5 = \theta_f \quad (4)$$

$$c_1 + 2c_2t_f + 3c_3t_f^2 + 4c_4t_f^3 + 5c_5t_f^4 = \dot{\theta}_f \quad (5)$$

$$2c_2 + 6c_3t_f + 12c_4t_f^2 + 20c_5t_f^3 = \ddot{\theta}_f \quad (6)$$

- Thus we will get these 6 equations. So there are in all 3 equations and 3 unknowns which need to be solved to get the value of the unknowns.

Solving above equations, We get

$$C_0 = \theta_i,$$

$$C_1 = \dot{\theta}_i,$$

$$C_2 = \frac{1}{2}\ddot{\theta}_i$$

$$C_3 = \frac{20(\theta_f - \theta_i) - (8\dot{\theta}_f + 12\dot{\theta}_i)t_f - (3\ddot{\theta}_i - \ddot{\theta}_f)t_f^2}{2t_f^3}$$

$$C_4 = \frac{30(\theta_i - \theta_f) + (14\dot{\theta}_f + 16\dot{\theta}_i)t_f + (3\ddot{\theta}_i - 2\ddot{\theta}_f)t_f^2}{2t_f^4}$$

$$C_5 = \frac{12(\theta_f - \theta_i) - 6(\dot{\theta}_f + \dot{\theta}_i)t_f - (\ddot{\theta}_i - \ddot{\theta}_f)t_f^2}{2t_f^5}$$

Numerical

A Numerical Example

- A single-link robot with a revolute joint is motionless at θ = 20°. It is desired to move the joint in a smooth manner to θ = 80° in 4.0 seconds. Find a suitable cubic polynomial to generate this motion and bring the manipulator to rest at the goal.

Solution:

cubic polynomial

$$\theta(t) = C_0 + C_1 t + C_2 t^2 + C_3 t^3 \quad \text{-------} \ (1)$$

Conditions:

At time $t = t_i = 0,$

$\theta = \theta_i = 20^\circ,$

$\dot{\theta} = 0;$

At time $t = t_f = 4.0$ s,

$\theta = \theta_f = 80^\circ,$

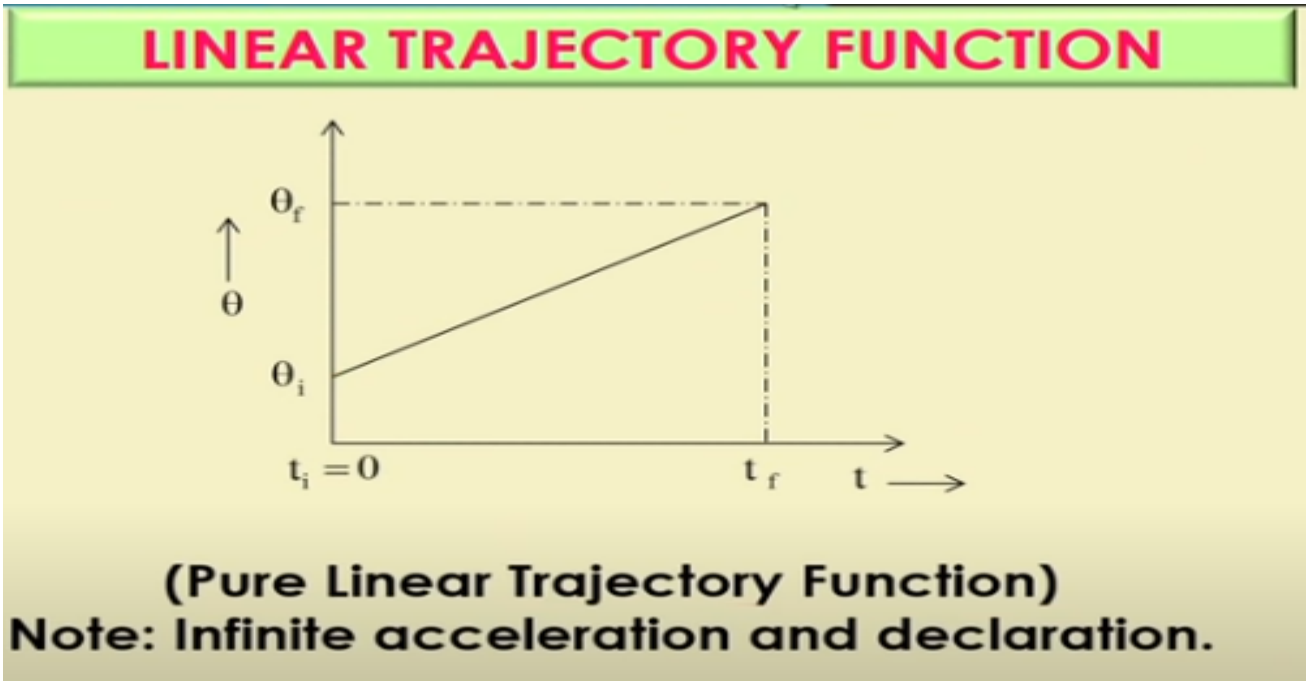
$\dot{\theta} = 0;$

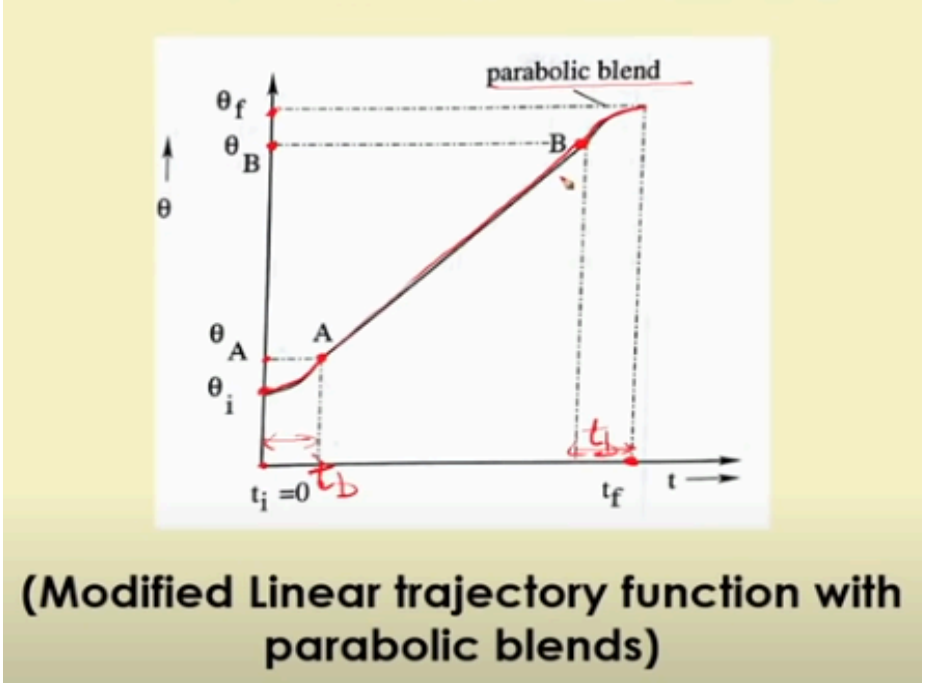
$$\theta(t) = \theta_i + \frac{3(\theta_f - \theta_i)}{t_f^2} t^2 - \frac{2(\theta_f - \theta_i)}{t_f^3} t^3$$

$$= 20 + \frac{3(80-20)}{(4.0)^2} t^2 - \frac{2(80-20)}{(4.0)^3} t^3$$

$$= 20 + 11.25t^2 - 1.875t^3$$

Linear Trajectory Function





(Modified Linear trajectory function with parabolic blends)

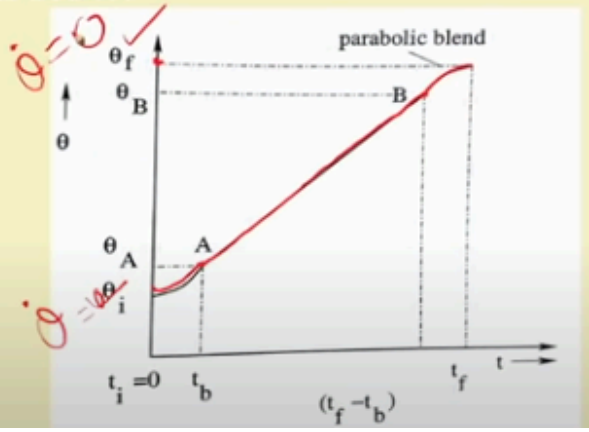
- **To avoid jerky movements at starting and end, we apply parabolic blends** which enable smooth motion of a robot. This eliminates infinite acceleration and deceleration.
- Here, the parabolic blends used are 2nd order curves.
- Now, let us solve a **numerical example** to better understand Numerical Trajectory function:

Numerical

Question

A Numerical Example

- A linear trajectory function with parabolic blends at its two ends is to be obtained to satisfy the following conditions given below.



At time $t = t_i = 0$,

$\theta = \theta_i = 20.0^o$,

$\dot{\theta} = \dot{\theta}_i = 0.0$;

At time $t = t_f = 12.0$ s,

$\theta = \theta_f = 74.0^o$,

$\dot{\theta} = \dot{\theta}_f = 0.0$;

Total cycle time $t_c = t_f - t_i = 12.0$ s

Time duration at each of the blend portion $t_b = 3.0$ s

Magnitude of acceleration/deceleration

$\ddot{\theta} = \alpha = 2.0$ degree/s^2

Determine angular displacement and velocity at two junctions of parabolic blends with the straight portion of trajectory function.

Solution

Solution:

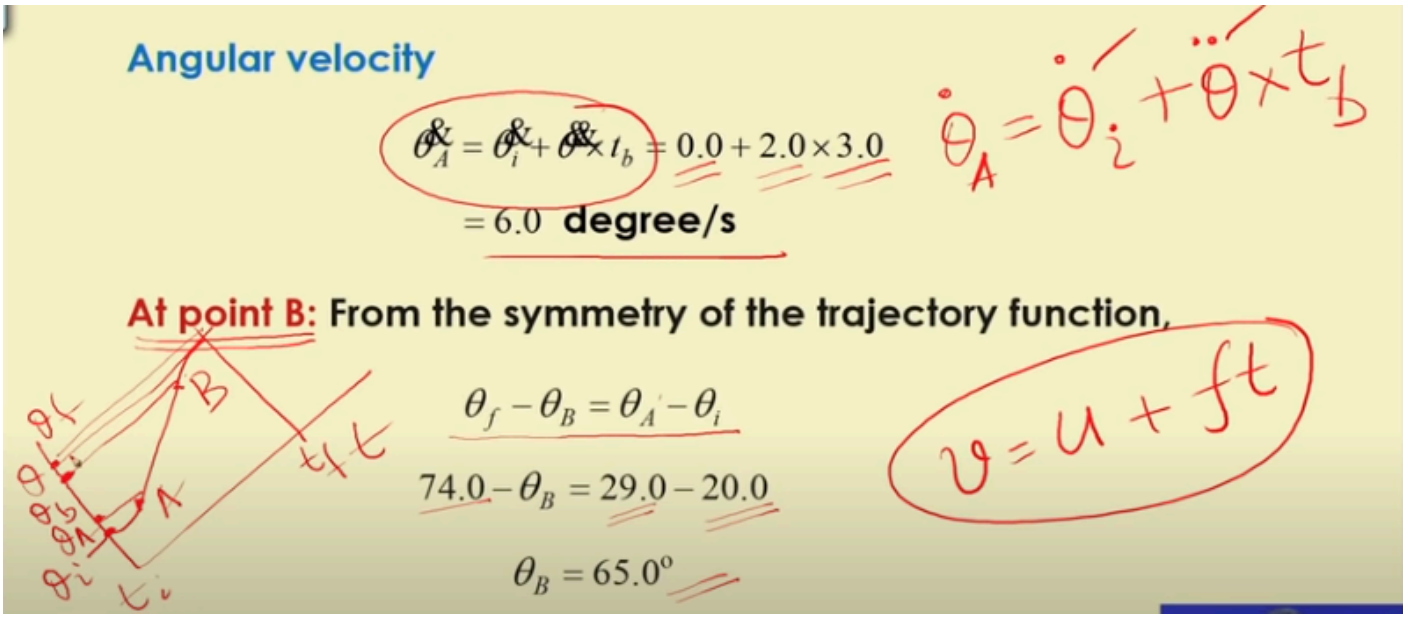
At point A

Angular displacement

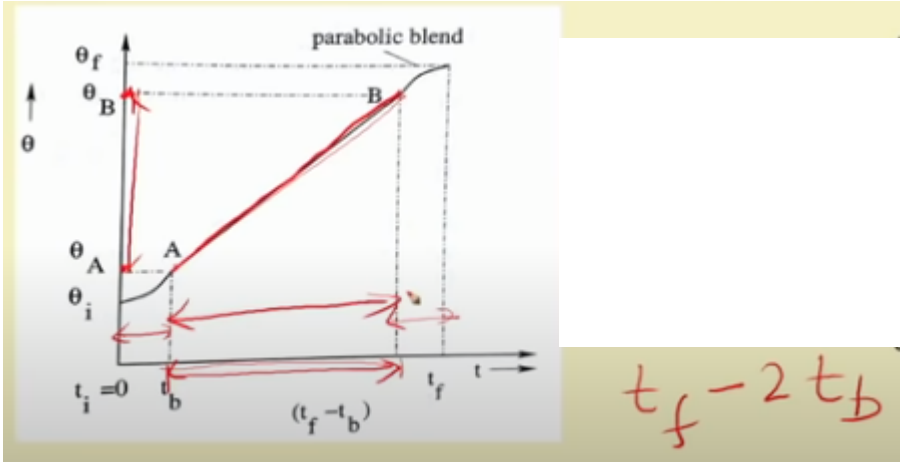
$$\theta_A = \theta_i + \frac{1}{2} \times \ddot{\theta} \times t_b^2 = 20.0 + \frac{1}{2} \times (2.0) \times (3.0)^2$$

$$= 29.0^o$$

- Firstly, the attempt is to find the Angular displacement at point A using a well known old formula learnt at school level of Angular displacement **S = ut + 1/2ft^2**
- Substituting the known values from the question to find **θ_A**



- Secondly, we have tried to determine the Angular velocity at Point A ($\dot{\theta}$) using an old formula learnt at school level of Angular Velocity, **v = u +ft**
- Now, at point B which is the junction of linear trajector and parabolic blend, the symmetry can be established by the relation $\theta_f - \theta_B = \theta_A - \theta_i$ (Refer to the small graph on the LHS to better understand it).
- So angular velocity at Point B is given by:



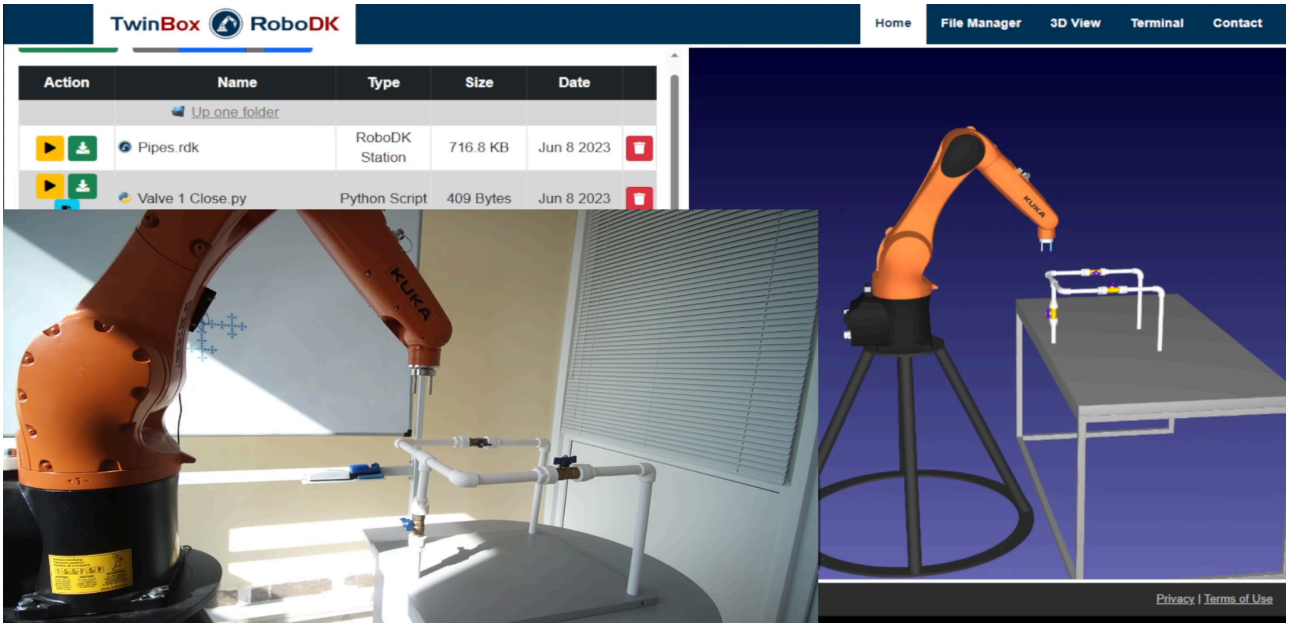
Angular velocity in the linear portion of trajectory function

$$= \frac{\theta_B - \theta_A}{t_f - 2t_b}$$

$$= \frac{65.0 - 29.0}{12.0 - 2 \times 3.0} = \frac{36.0}{6.0} = 6.0 \text{ degree/s}$$

To maintain continuity of the trajectory function at point B, $\dot{\theta}_B$ Should be equal to the velocity of the linear portion, that is, 6.0 degree per second.

Dynamic Simulation in Robotics: A Virtual Playground for Robot Testing



In robotics, a **dynamic simulation** is a computer-generated model that imitates the **realistic behavior** of a robot in a virtual environment. Unlike kinematic simulations, which only consider the geometric relationships between joints and the end-effector's position, dynamic simulations incorporate the **forces and torques** acting on the robot. This allows for a more accurate representation of how the robot will **move, interact with its surroundings, and respond to external influences**.

Key Components of a Dynamic Simulation:

- **Robot Model:** This is a mathematical representation of the robot's physical properties, including:
  - **Mass and inertia** of different robot links.
  - **Joint limits and constraints**.
  - **Kinematic structure** (how joints are connected).
- **Dynamic Engine:** This software component solves the **equations of motion** based on the robot model and the applied forces and torques. These equations relate the forces acting on the robot to its resulting accelerations and movements.
- **Simulation Environment:** This provides a virtual world with objects and scenarios for the robot to interact with. The environment can be tailored to represent real-world conditions, including:
  - **Gravity**.
  - **Friction**.

- **Obstacles**.
- **Other robots or moving objects**.

Benefits of Dynamic Simulation:

- **Testing and Validation:** Safely test robot control algorithms, path planning strategies, and identify potential issues before deploying the robot in the real world. This helps avoid costly mistakes and damage to physical hardware.
- **Performance Analysis:** Evaluate robot performance under various conditions (e.g., different loads, environmental factors) and optimize robot design. Simulations allow you to test a wide range of scenarios without the need for expensive physical prototypes.
- **Training and Education:** Provide a safe environment for training operators and programmers on robot behavior and control. Users can experiment with different settings and strategies without risk to the physical robot or its surroundings.
- **Reduced Development Costs:** By testing and troubleshooting in a virtual environment, development cycles can be accelerated, and resources can be saved compared to physical testing.

Applications of Dynamic Simulation:

- **Robot Control Design:** Simulate different control algorithms to ensure smooth, precise, and efficient robot movements.
- **Offline Programming:** Develop and test robot programs in a safe virtual environment before deployment.
- **Human-Robot Collaboration:** Simulate scenarios where robots interact with humans to assess potential safety hazards and optimize interaction strategies.
- **Robot-Assisted Surgery:** Train surgeons on robotic surgical procedures in a risk-free virtual environment.

Limitations of Dynamic Simluation:

- **Model Accuracy:** The accuracy of the simulation depends on the quality of the robot model and the complexity of the environment being simulated.
- **Computational Cost:** Complex simulations with many degrees of freedom and intricate environments can be computationally expensive and require powerful computers.
- **Real-world Uncertainties:** While simulations can be very detailed, they may not capture all the complexities and uncertainties of the real world, such as sensor noise or unexpected environmental conditions.

Overall, dynamic simulation is a powerful tool for engineers, researchers, and developers working in robotics. By creating realistic virtual models, they can test, analyze, and optimize robot behavior before real-world deployment. This leads to:

- **Improved robot design:** By simulating the effects of forces and torques, engineers can design robots that are more efficient, robust, and capable of handling different environments.
- **Enhanced robot control:** Dynamic simulations help develop and refine control algorithms that account for the complex dynamics of robot motion.
- **Reduced development costs:** Testing and troubleshooting in a virtual environment allows for faster development cycles and saves resources compared to physical testing.

Robot Simulation Softwares/ Environment
(only for information, not included in the syllabus)

General-Purpose Multiphysics Simulation Platforms:

- **Ansys**: A comprehensive suite of engineering simulation software, including modules for structural analysis, fluid dynamics, and multibody dynamics. Ansys offers features like robot module integration, customizable material properties, and powerful post-processing capabilities.
- **Abaqus**: Another versatile engineering simulation platform with strong capabilities in finite element analysis (FEA) and multibody dynamics. Abaqus excels in simulating complex contact mechanics and material behavior relevant to robots interacting with their environment.
- **COMSOL Multiphysics**: A powerful platform for modeling and simulating a wide range of physical phenomena, including mechanics, electromagnetism, and fluid dynamics. COMSOL offers a user-friendly interface and customizable libraries for building robot models and simulating their interactions with various environments.

Robot-Specific Simulation Software:

- **Adams**: A dedicated multibody dynamics software specifically designed for simulating mechanical systems, including robots. Adams offers a rich library of robot components, user-friendly tools for building robot models, and features for simulating contact forces and actuator dynamics.
- **RoboDK**: A powerful robot simulation software with integrated path planning and programming capabilities. RoboDK allows users to import CAD models of robots and their workspaces, create realistic simulations, and even generate robot programs directly within the simulation environment.
- **V-REP**: An open-source robot simulation platform offering a flexible and customizable environment for building complex robot models and scenarios. V-REP provides features for simulating sensors, actuators, and complex interactions between robots and their surroundings.

Additional Options:

- **Gazebo**: A robot simulation platform from Open Robotics, known for its integration with the Robot Operating System (ROS). Gazebo offers a rich physics engine, sensor simulation capabilities, and support for various robot models, making it a popular choice for researchers developing robot control algorithms.
- **MuJoCo**: A physics engine specifically designed for simulating legged robots and other complex underactuated systems. MuJoCo is known for its fast and efficient simulation capabilities, making it suitable for real-time control applications.

Choosing the right software depends on several factors, including:

- **Complexity of the robot model:** Some software may be better suited for simpler robots, while others can handle highly complex models with many degrees of freedom.
- **Desired level of realism:** Different software offers varying levels of detail in simulating physics, contact mechanics, and sensor dynamics.
- **Budget:** Commercial software licenses can be expensive, while open-source options offer free access but may require more programming expertise.
- **Integration with other tools:** Consider compatibility with your existing development tools, CAD software, or control algorithms.

Computational Considerations

In the context of dynamic simulation for robotics, computational considerations refer to the factors that affect the **processing power and time** required to run a simulation effectively. These considerations are crucial for achieving accurate and useful results while efficiently utilizing computing resources.

Computational considerations in dynamic simulation:

- **Robot Model Complexity:**
  - **Degrees of Freedom (DOF):** Robots with more joints and DOF lead to a larger number of equations that need to be solved during the simulation, increasing computational demands.
  - **Model Detail:** Complex models with intricate geometries, material properties, and contact interactions require more calculations compared to simpler models.
- **Simulation Environment Complexity:**
  - **Number of Objects:** Simulating a robot interacting with numerous objects in the environment increases the complexity of collision detection and contact mechanics calculations.
  - **Environmental Factors:** Simulating effects like wind, friction, and fluid dynamics adds another layer of complexity and computational cost.
- **Simulation Time Step:**
  - **Accuracy vs. Speed:** Smaller time steps provide higher accuracy by capturing faster dynamics but require more calculations per unit time.
  - **Trade-off:** Finding the optimal time step balance between accuracy and computational efficiency is crucial.
- **Numerical Methods:**
  - **Solver Algorithms:** Different numerical methods used to solve the equations of motion can have varying levels of computational complexity.

Some methods may be more efficient for specific types of robot models or simulation scenarios.

- **Hardware Capabilities:**
  - **Processing Power:** The processing power (CPU cores and clock speed) of your computer directly impacts how quickly the simulation can be run.
  - **Memory (RAM):** The amount of RAM available affects the simulation's ability to store intermediate calculations and complex model data.

Strategies for Managing Computational Cost:

- **Model Simplification:** If possible, simplify the robot model by reducing unnecessary detail or using lumped mass approximations without compromising the desired level of accuracy.
- **Environment Optimization:** Limit the number of objects and environmental factors in the simulation to focus on the most critical aspects relevant to the analysis.
- **Adaptive Time Stepping:** Utilize algorithms that adjust the time step size dynamically based on the complexity of the simulated scene, focusing on smaller steps during critical high-speed interactions.
- **Efficient Numerical Methods:** Explore alternative numerical methods known to be efficient for specific robot and simulation types.
- **Parallelization:** Leverage multi-core processors or high-performance computing clusters to distribute the computational load and achieve faster simulations.

By carefully considering these computational considerations, researchers and engineers can optimize their dynamic simulations for robots. This ensures efficient use of computing resources while achieving the desired level of accuracy and realism in the simulated robot behavior.

Lagrangian Formulation of Manipulator Dynamics:

Introduction:

The dynamics of robotic manipulators, such as robotic arms, play a crucial role in understanding how these systems move and interact with their environment. The study of manipulator dynamics involves predicting the motion of the manipulator under the influence of external forces and torques.

Need for Lagrangian Formulation:

When it comes to analyzing the dynamics of robotic manipulators, various methods can be employed. One powerful approach is the Lagrangian formulation. This method offers a systematic and elegant way to derive the equations of motion for complex mechanical systems, including robotic manipulators.

Reasons for Choosing Lagrangian Formulation:

1. Systematic Approach: The Lagrangian formulation provides a systematic framework for deriving equations of motion by considering the energy of the system.
2. Holistic Representation: It allows us to represent the entire system's dynamics in terms of a single function, the Lagrangian, which encapsulates both kinetic and potential energy.
3. Generalization: The use of generalized coordinates in Lagrangian mechanics allows for a more generalized treatment of systems with varying degrees of freedom.
4. Conservation Laws: Lagrangian mechanics naturally incorporates conservation laws, such as the conservation of energy and momentum, simplifying the analysis.
5. Flexible for Complex Systems: This approach is particularly useful for manipulators with multiple links and joints, as it handles complex configurations and constraints efficiently.

By employing the Lagrangian formulation, we can derive the equations of motion for robotic manipulators, enabling us to predict their behavior accurately and design control strategies for various tasks. Now, let's understand the mathematical derivation of the Lagrangian formulation for manipulator dynamics.

Tensors:

- Mathematical objects that capture geometric and physical properties in a coordinate-independent way.
- They can be used to represent quantities like:
    - **Inertia matrix (M):** This matrix appears in the kinetic energy expression and depends on the robot's link lengths, masses, and configuration (joint angles). The inertia matrix can be expressed using inertia tensors associated with each link.
    - **Geometric properties:** Denavit-Hartenberg (DH) parameters, which define the robot's geometry, can be viewed as components of tensors.

These parameters are used to calculate positions and velocities of manipulator links.

Connection:

- While the Lagrangian itself is a scalar, the derivation of the equations of motion using Lagrange's equations involves manipulating expressions with tensors.
- The kinetic energy expression often includes the inertia matrix (M), which can be derived from the manipulator's link properties using inertia tensors.
- DH parameters, which are components of tensors, are used to calculate link positions and velocities, which are then used to find the kinetic energy.

In essence, tensors provide a powerful tool for representing the geometric and physical properties of the robot that are then used to build the expressions involved in the Lagrangian formulation.

Here's an analogy: Imagine a recipe (Lagrangian) for baking a cake (robot motion). The recipe itself might just list ingredients (scalar quantities), but it relies on understanding the shapes and sizes of pans (tensors) and the properties of the ingredients (also tensors) to achieve the final result.

The Newton–Euler approach is based on the elementary dynamic formulas (6.29) and (6.30) and on an analysis of forces and moments of constraint acting between the links. As an alternative to the Newton–Euler method, in this section we briefly introduce the **Lagrangian dynamic formulation**. Whereas the Newton–Euler formulation might be said to be a "force balance" approach to dynamics, the Lagrangian formulation is an "energy-based" approach to dynamics. Of course, for the same manipulator, both will give the same equations of motion. Our statement of Lagrangian dynamics will be brief and somewhat specialized to the case of a serial-chain mechanical manipulator with rigid links.

We start by developing an expression for the kinetic energy of a manipulator. The kinetic energy of the ith link, k_i, can be expressed as

$$k_i = \frac{1}{2} m_i v_{C_i}^T v_{C_i} + \frac{1}{2} {}^i\omega_i^T \, {}^{C_i} I_i \, {}^i\omega_i,$$

where the first term is kinetic energy due to linear velocity of the link's center of mass and the second term is kinetic energy due to angular velocity of the link. The total kinetic energy of the manipulator is the sum of the kinetic energy in the individual links—that is,

$$k = \sum_{i=1}^{n} k_i.$$

The v_{C_i} and ${}^{i}\omega_i$ in (6.69) are functions of Θ and $\dot{\Theta}$, so we see that the kinetic energy of a manipulator can be described by a scalar formula as a function of joint position and velocity, $k(\Theta, \dot{\Theta})$. In fact, the kinetic energy of a manipulator is given by

$$k(\Theta, \dot{\Theta}) = \tfrac{1}{2}\dot{\Theta}^T M(\Theta)\dot{\Theta},$$

where $M(\Theta)$ is the $n \times n$ manipulator mass matrix already introduced in Section 6.8. An expression of the form of (6.71) is known as a **quadratic form** [5], since when expanded out, the resulting scalar equation is composed solely of terms whose dependence on the $\dot{\theta}_i$ is quadratic. Further, because the total kinetic energy must always be positive, the manipulator mass matrix must be a so-called **positive definite** matrix. Positive definite matrices are those having the property that their quadratic form is always a positive scalar. Equation (6.71) can be seen to be analogous to the familiar expression for the kinetic energy of a point mass:

$$k = \tfrac{1}{2}mv^2.$$

The fact that a manipulator mass matrix must be positive definite is analogous to the fact that a scalar mass is always a positive number.

The potential energy of the ith link, u_i, can be expressed as

$$u_i = -m_i\, {}^{0}g^T\, {}^{0}P_{C_i} + u_{ref_i},$$

where ${}^{0}g$ is the 3×1 gravity vector, ${}^{0}P_{C_i}$ is the vector locating the center of mass of the ith link, and u_{ref_i} is a constant chosen so that the minimum value of u_i is zero.[1] The total potential energy stored in the manipulator is the sum of the potential energy in the individual links—that is,

$$u = \sum_{i=1}^{n} u_i.$$

Because the ${}^{0}P_{C_i}$ in (6.73) are functions of Θ, we see that the potential energy of a manipulator can be described by a scalar formula as a function of joint position, $u(\Theta)$.

The Lagrangian dynamic formulation provides a means of deriving the equations of motion from a scalar function called the **Lagrangian**, which is defined as the difference between the kinetic and potential energy of a mechanical system. In our notation, the Lagrangian of a manipulator is

$$\mathcal{L}(\Theta, \dot{\Theta}) = k(\Theta, \dot{\Theta}) - u(\Theta).$$

The equations of motion for the manipulator are then given by

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\Theta}} - \frac{\partial \mathcal{L}}{\partial \Theta} = \tau,$$

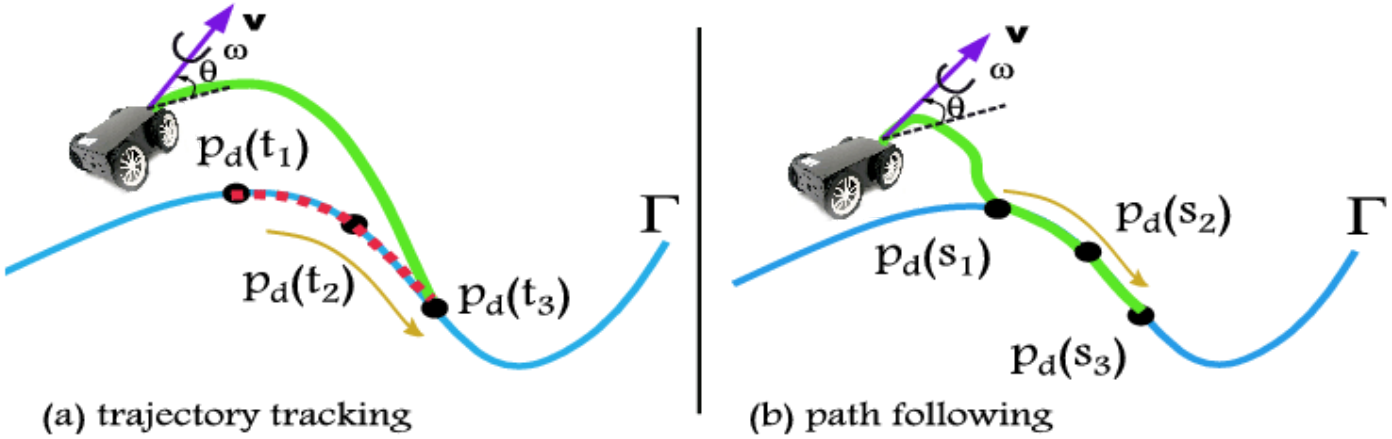
where τ is the $n \times 1$ vector of actuator torques. In the case of a manipulator, this equation becomes

$$\frac{d}{dt}\frac{\partial k}{\partial \dot{\Theta}} - \frac{\partial k}{\partial \Theta} + \frac{\partial u}{\partial \Theta} = \tau,$$

where the arguments of $k(\cdot)$ and $u(\cdot)$ have been dropped for brevity.

Unit 4: Motion Control

Path vs. Trajectory in Robotics



In the world of robotics, achieving precise and efficient robot motion requires careful planning. Understanding the distinction between **path** and **trajectory** is fundamental to this planning process. While both concepts relate to robot movement, they represent different levels of detail:

Path:

- **Focus:** The **geometric blueprint** of the robot's movement, defining the sequence of positions the end-effector (gripper or tool) needs to reach.
- **Information:**
  - A series of **waypoints**, which are the specific locations the end-effector must visit.
  - May include **intermediate points** along a curved path or just the start and end positions for straight-line movements.
- **Time Independence: Doesn't specify** the timing or speed at which the robot travels between points. It's like a roadmap that shows the route but doesn't tell you how long it takes to travel each segment.
- **Applications:**
  - Planning the overall route for a robot arm to assemble parts.
  - Defining waypoints for a mobile robot to navigate an obstacle course.

Trajectory:

- **Focus:** The **complete picture** of the robot's movement, encompassing both **position and timing**.

- **Information:**
  - Defines the **position and orientation** of the end-effector as a function of **time**.
  - Specifies not only where the robot needs to be, but also **when** and **how fast** it should get there.
- **Time Dependence:** Considers the **timing** for reaching waypoints and the desired **speed profile** (velocity) at each point along the path. It's like the roadmap with markers indicating travel times between points and suggested speeds for different sections.
- **Applications:**
  - Generating a smooth and efficient path for a robot arm to pick and place an object, specifying not just the positions but also the speed at which the arm should move at different stages to avoid jerky motions.
  - Controlling a mobile robot to follow a path while maintaining a specific orientation or adhering to speed limits in different zones.

Analogy: Think of a car trip. The **path** is the chosen route (e.g., highway, scenic route), while the **trajectory** would include specific details like the planned departure time, estimated arrival time, and desired speed limits along different parts of the journey.

Planning Strategies:

- **Path Planning:**
  - Focuses on finding a collision-free path that satisfies task constraints while optimizing factors like distance, travel time, or energy consumption.
  - Various algorithms exist for path planning, such as Dijkstra's algorithm for finding the shortest path or A* search for balancing path length and other criteria.
- **Trajectory Planning:**
  - Builds upon the defined path by incorporating timing and velocity profiles.
  - Considers robot dynamics (forces and torques) to ensure the robot can physically follow the planned path with its specified speeds and accelerations.
  - Trajectory planning techniques often involve solving the equations of motion for the robot or employing interpolation methods between waypoints.

Importance of Path and Trajectory:

- **Accuracy and Efficiency:** Precise path planning ensures the robot reaches its target location, while efficient path planning minimizes unnecessary movements and travel time.
- **Collision Avoidance:** Paths are planned to avoid obstacles, ensuring safe robot operation.
- **Smooth Motion:** Trajectory planning generates smooth and continuous paths, minimizing jerks and vibrations that could damage the robot or the environment.
- **Task Completion:** Proper planning ensures the robot can complete its tasks successfully.

Overall, understanding the distinction between path and trajectory is essential for effective robot control. Careful path planning defines the desired route, while trajectory planning ensures the robot follows that path in a smooth, efficient, and safe manner. Both planning techniques work in tandem to achieve precise and controlled robot movements in various robotic applications.

Feature	Path	Trajectory
Focus	Geometric layout of movement	Complete picture of movement (position and timing)
Information	Sequence of waypoints (positions)	Position and orientation as a function of time
Time Dependence	No	Yes, specifies timing and speed profile
Example	Planning a route on a map	Planning a smooth path with travel times

Joint Space versus Cartesian Space Descriptions

In robotics, there are two primary ways to describe the motion and configuration of a robot manipulator: **joint space** and **Cartesian space**. Each offers a different perspective on how the robot moves and interacts with its environment.

Joint Space:

- **Focus:** Focuses on the **individual joint angles** or positions of the robot's joints.
- **Information:**

- Represents the robot's configuration using a set of values, one for each joint angle ($\theta 1, \theta 2, ..., \theta n$).
- These angles define the relative positions of the robot's links with respect to each other.

- **Advantages:**
  - Simple and intuitive for robot control, especially for low-level motor commands.
  - Many robot control algorithms are designed and implemented in joint space.
  - Easier to perform inverse kinematics calculations (solving for joint angles to achieve a desired end-effector pose).
- **Disadvantages:**
  - Can be complex to plan motions for tasks that require specific end-effector positions or orientations in the workspace (Cartesian space).
  - Doesn't directly reflect the robot's interaction with its environment.

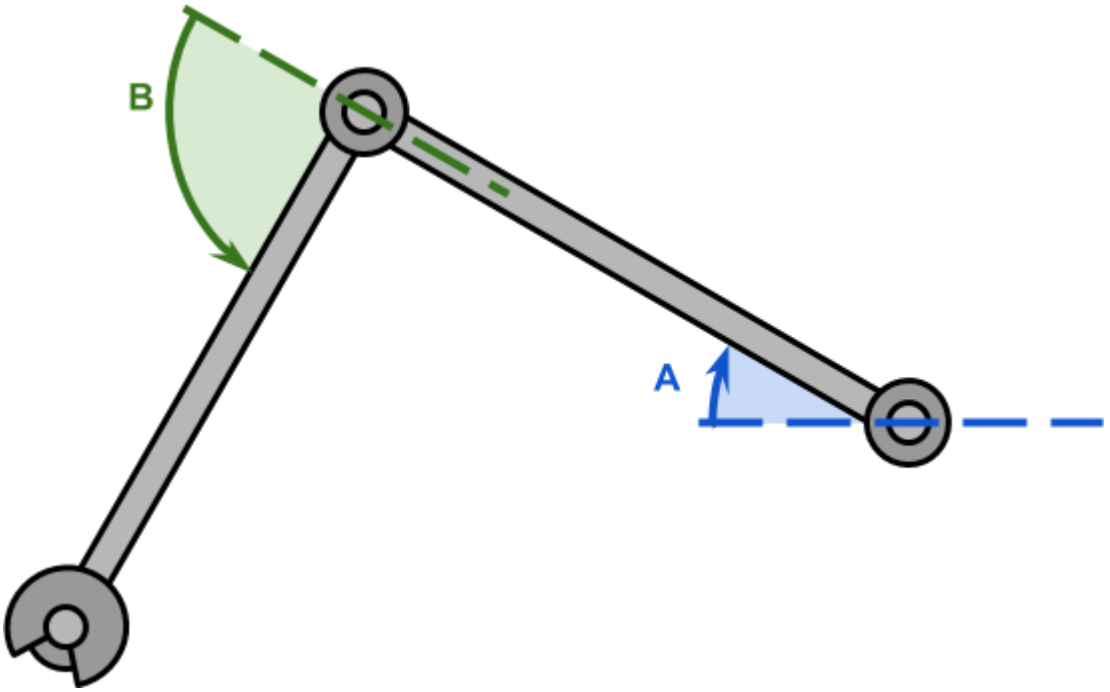
Cartesian Space (Operational Space):

- **Focus:** Focuses on the position and orientation of the robot's **end-effector** (gripper or tool) relative to a reference frame.
- **Information:**
  - Described using a set of coordinates (X, Y, Z) for the end-effector's position and additional parameters (Euler angles, quaternions, or rotation matrices) to represent its orientation.
- **Advantages:**
  - More intuitive for humans to understand and plan robot motions, as it directly relates to the task being performed.
  - Easier to specify desired end-effector poses for tasks like grasping objects or navigating in the environment.
- **Disadvantages:**
  - Forward kinematics calculations (solving for joint angles to achieve a desired end-effector pose) can be more complex, especially for robots with many degrees of freedom.
  - May require additional processing to translate desired Cartesian paths into joint space commands for the robot.

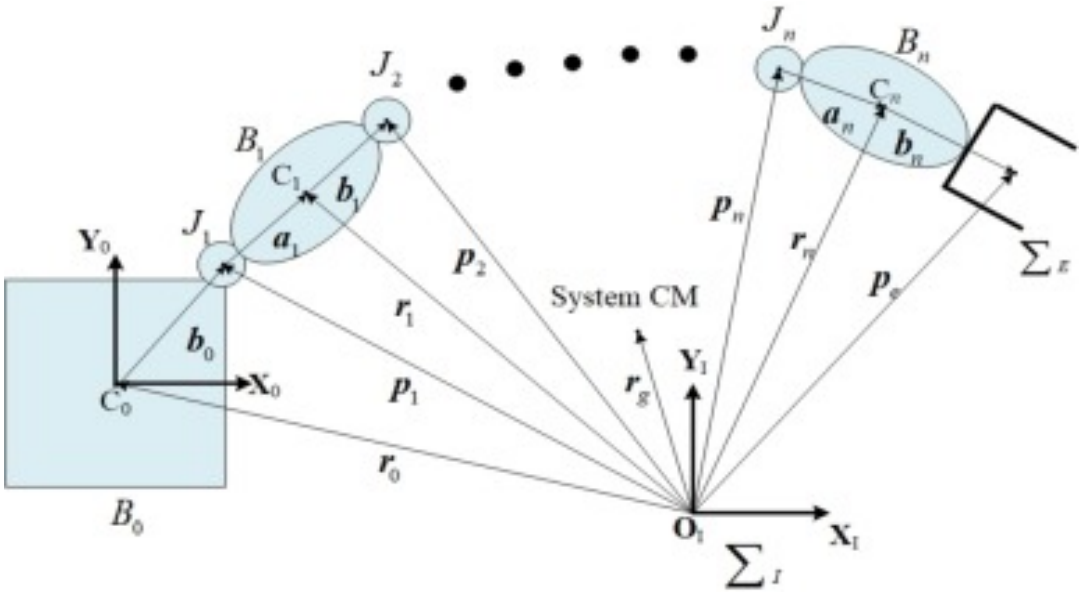
Choosing the Right Space:

The choice between joint space and Cartesian space for robot control depends on the specific application and priorities:

- **Joint space** is often preferred for:
  - Low-level control of individual joints.
  - Applications where joint angles are directly relevant to the task (e.g., joint-based manipulation tasks).



- **Cartesian space** is often preferred for:
  - Task-level planning and programming, as it allows for easier specification of desired end-effector poses.
  - Applications where the robot interacts with the environment in a specific way (e.g., object grasping, path following).



Combining Both Approaches:

In many cases, a combination of both joint space and Cartesian space approaches is used. For instance:

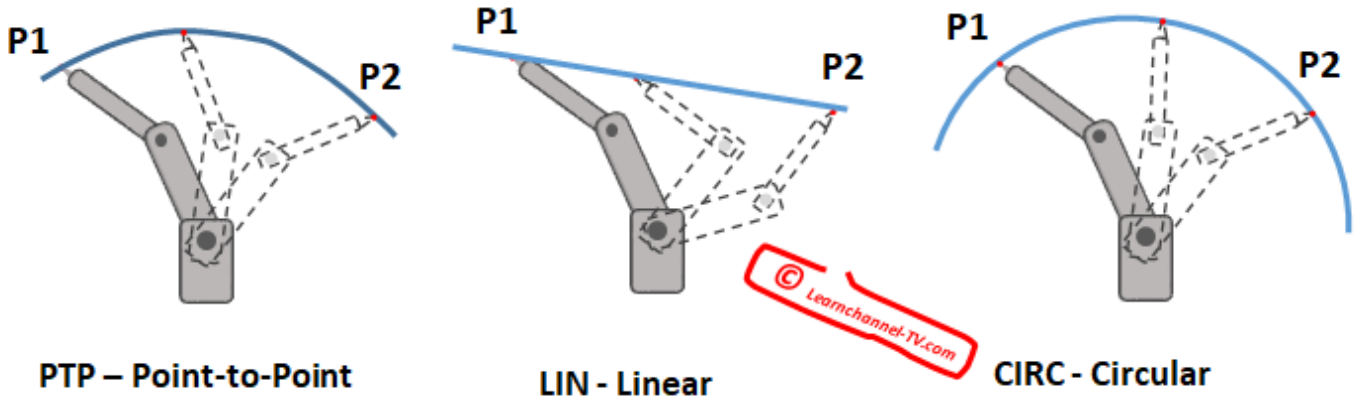
- **Motion planning** can be done in Cartesian space to define the desired end-effector path.
- **Inverse kinematics** can then be used to translate the desired path into a sequence of joint space commands for the robot to follow.

Additional Considerations:

- **Robot Kinematics:** The relationship between joint angles and end-effector pose is crucial for both joint space and Cartesian space control.
- **Robot Dynamics:** Forces and torques acting on the robot need to be considered when translating between joint space and Cartesian space for real-world motion execution.

Feature	Joint Space	Cartesian Space (Operational Space)
Focus	Individual joint angles (θ1, θ2, ..., θn)	End-effector position and orientation
Information	Set of joint angles representing relative link positions	X, Y, Z coordinates for position; Euler angles, quaternions, or rotation matrices for orientation
Advantages	* Simple and intuitive for low-level control * Easier inverse kinematics calculations	* Easier to plan tasks in workspace * More intuitive for humans
Disadvantages	* Complex for Cartesian space task planning * Doesn't reflect interaction with environment	* Forward kinematics can be complex * Requires translation to joint space commands
Applications	* Low-level motor commands * Joint-based manipulation tasks	* Task-level planning and programming * Object grasping, path following
Control Approach	Often used for direct control of robot joints	Often used for high-level task specification
Planning	May require converting desired Cartesian paths to joint space commands	Easier to define desired end-effector poses

Point-to-Point (PTP) Control in Robotics: Moving from A to B Efficiently



Point-to-point (PTP) control is a fundamental technique used in robotics to move a robot's end-effector (gripper or tool) from one **desired position** to another **desired position** in the workspace as quickly and efficiently as possible. It focuses on achieving the target positions without necessarily controlling the path the end-effector takes to get there.

Key Characteristics:

- Reach the target point precisely, not necessarily following a specific path between points.
- Requires the robot to know the **starting position** and the **desired target position** of the end-effector.
- **Control Strategy:**
    - The robot controller calculates the necessary joint angles or velocities to move from the starting point to the target point.
    - This may involve solving the inverse kinematics problem to determine the required joint angles for the desired end-effector pose.
    - The robot then moves each joint independently to reach its target position as quickly as possible while respecting joint limits and actuator capabilities.
- **Trajectory:** The actual path taken by the end-effector depends on the robot's kinematics and the chosen control strategy. It's not explicitly controlled in PTP, unlike continuous-path control.

Advantages:

- **Simplicity:** PTP control is relatively straightforward to implement, making it suitable for many basic robotic tasks.
- **Speed and Efficiency:** PTP control prioritizes reaching the target point quickly, making it efficient for tasks where precise path following is not critical.
- **Reduced Complexity:** By not focusing on the specific path, PTP control avoids the complexities of path planning and trajectory generation.

Disadvantages:

- **Non-optimized Paths:** The robot may take inefficient paths between points, leading to unnecessary movements and potentially higher energy consumption.
- **Jerky Motions:** Rapid changes in joint velocities can cause jerky and abrupt movements, which might be undesirable for delicate tasks or tasks involving fragile objects.
- **Limited Applications:** PTP control might not be suitable for tasks requiring smooth and controlled movements along a specific path, such as painting or welding.

Applications:

- **Pick-and-place operations:** Moving a robot arm to pick up an object from one location and place it in another.
- **Material handling:** Transferring objects from one point to another in a warehouse or assembly line.
- **Machine loading and unloading:** Precisely positioning objects for machining or other processes.
- **Simple tasks** where precise path following is not critical and speed is a priority.

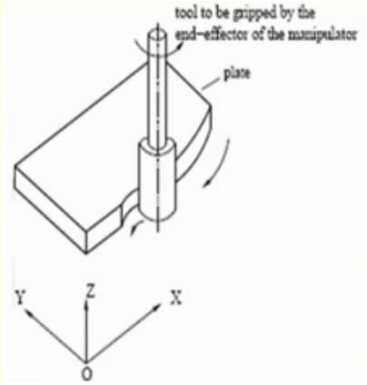
Continuous Path Control

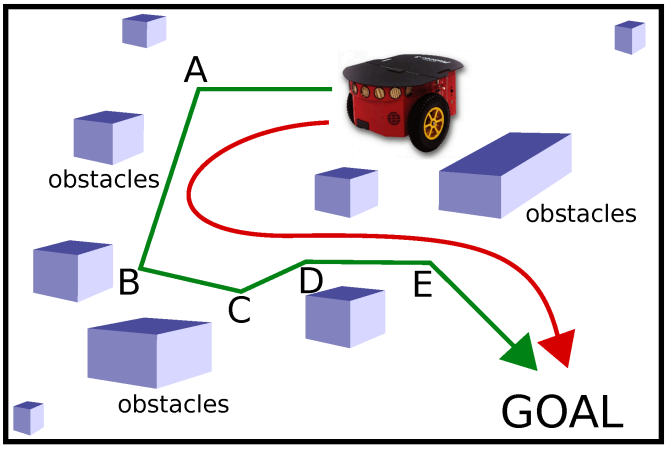
Imagine you're making a peanut butter and jelly sandwich. Here's how point-to-point (PTP) and continuous-path (CP) control come into play with a robotic arm:

Point-to-Point (PTP) Control:

1. **Grab the Bread:** The robot arm quickly moves its end-effector (gripper) to the exact position of the bread slice, like a fast and efficient point-to-point jump. It doesn't care about the exact path it takes to get there.

2. **Spread the Peanut Butter:** The robot might move its gripper in a jerky way (not smooth) as it positions itself at different points to spread the peanut butter evenly. This jerky motion could potentially rip the bread!





Continuous-Path (CP) Control:

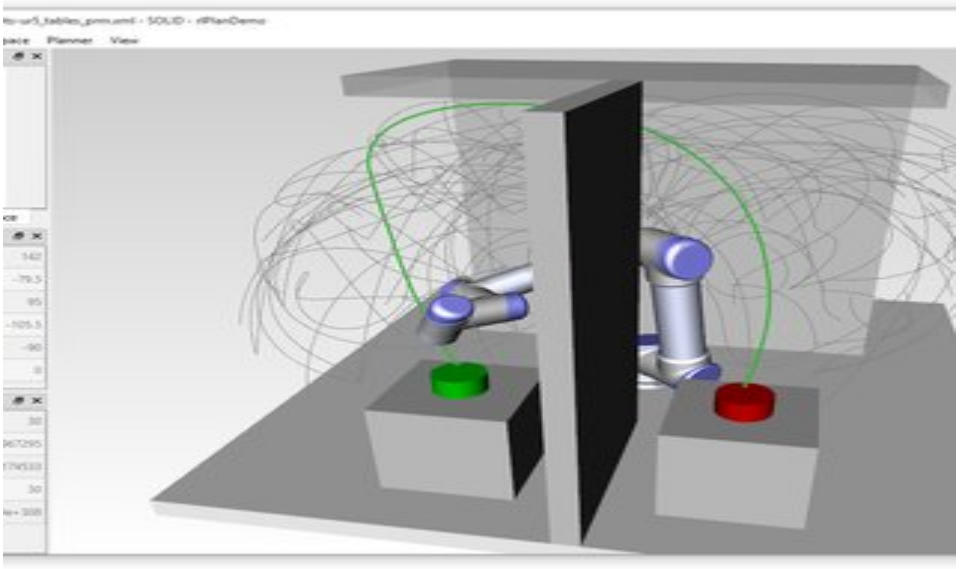
1. **Grab the Bread:** Similar to PTP, the robot moves to the bread, but...
2. **Spread the Peanut Butter:** Now, imagine the robot's gripper with the peanut butter jar follows a smooth, controlled path across the bread like a person spreading it with a knife. The robot controls not just the final position but also the entire smooth path of the gripper, ensuring an even and delicious spread without ripping the bread.

Key Differences:

- **PTP:** Like quickly pointing to different spots on a map, it focuses on reaching specific points (bread and jar) without a smooth path in between.
- **CP:** Like following a drawn line on a map, it controls the entire smooth path the robot takes to complete the task (spreading).

In simpler terms, PTP control is like quickly jumping from point A to point B, while CP control is like smoothly following a path to get from A to B. For tasks requiring smooth and precise movements, CP control is the better choice!

Trajectory Generation



Trajectory generation in robotics is like planning a detailed route for your robot to follow, but instead of a car trip, it considers the robot's movements. Here's a breakdown using a simple example:

Imagine you have a robot arm that needs to pick up a block and place it in a specific spot.

- **Without Trajectory Generation:** You might simply move the robot arm's joints however you can to reach the block and then move it again to place it. This could be jerky and imprecise, potentially knocking over the block!
- **With Trajectory Generation:** It's like planning a smooth and efficient route for the robot arm:
    1. **Start Position:** The robot knows its starting position.
    2. **Goal Position:** It knows where the block is and where it needs to be placed.
    3. **Planning the Path:** A computer program calculates the smoothest way for the robot arm to move its joints to reach the block without jerking or hitting anything.
    4. **Speed and Timing:** The program also determines the speed at which each joint should move at different points to ensure a smooth and controlled motion.
    5. **Following the Plan:** The robot arm follows the planned trajectory, moving its joints in a coordinated way to pick up the block and place it precisely.

Benefits of Trajectory Generation:

- **Smooth and Precise Movements:** The robot moves smoothly and avoids jerky motions, reducing the risk of damaging objects or itself.
- **Efficiency:** The planned path can be optimized for speed and energy consumption.
- **Accuracy:** The robot is more likely to reach the desired positions precisely.

Trajectory generation is like creating a detailed instruction manual for your robot, ensuring its movements are smooth, efficient, and accurate for various tasks.

Control Systems

Force Control

Force control in robotics is the ability of a robot to **sense and respond to external forces** acting on it while performing a task. It allows the robot to interact with its environment in a controlled and delicate manner, similar to how we use our sense of touch for tasks.

Imagine a robot arm trying to crack an egg.

- **Without Force Control:** The robot arm might simply press down on the egg with a predetermined force. This could easily crush the egg!
- **With Force Control:** The robot arm has a built-in force sensor in its gripper:
    1. **Sensing Force:** As the gripper touches the egg, the sensor detects the increasing force.
    2. **Adjusting Movement:** The robot program is designed to interpret the force sensor data. If the force becomes too high (indicating potential

crushing), the program instructs the robot to stop pressing or even slightly back away.

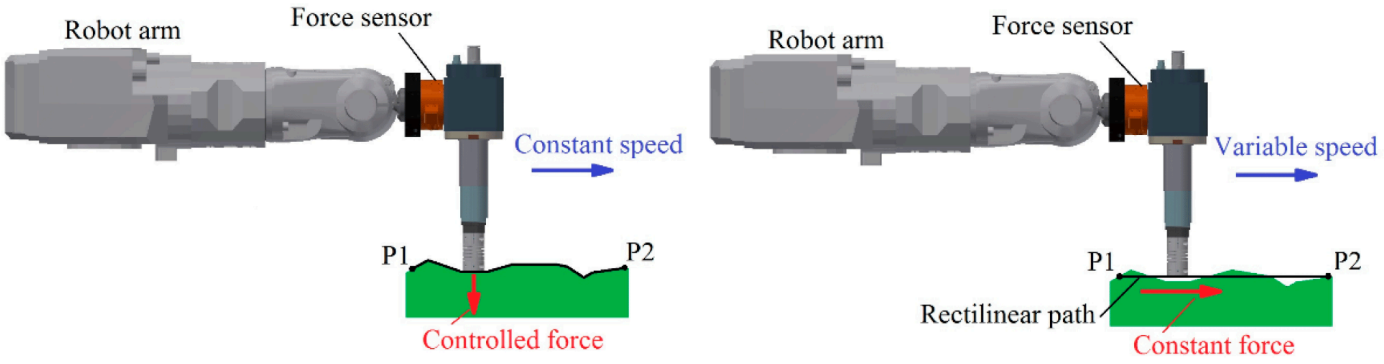
3. **Controlled Cracking:** The robot can apply a gentle and controlled force, just enough to crack the eggshell without damaging the yolk.

Benefits of Force Control:

- **Delicate Tasks:** Allows robots to handle fragile objects or perform tasks requiring precise force application (e.g., inserting components, polishing surfaces).
- **Improved Safety:** Reduces the risk of damaging objects or the robot itself by enabling real-time force adjustments.
- **Versatility:** Enables robots to interact with a wider range of environments and objects that may vary in shape, size, and texture.

Force control is like giving your robot a sense of touch. By sensing and responding to external forces, robots can perform tasks that require delicate manipulation and interaction with the environment.

Hybrid Position/ Force Control System



A hybrid position/force control system combines the best of both worlds: **precise positioning** and **sensitive force control**. Imagine you're a skilled sculptor using a robotic arm to create a masterpiece. Here's how this hybrid system would work:

- **Without Hybrid Control:**
    1. **Pure Position Control:** You might struggle to control the robot arm's position precisely, potentially damaging the sculpture with accidental bumps.

2. **Pure Force Control:** The arm might be too sensitive to the touch, constantly adjusting and never achieving a stable position for sculpting details.

- **With Hybrid Control:** The system seamlessly blends both controls:
  1. **Guiding the Tool:** You instruct the robot arm to move its end-effector (tool) to specific locations with some level of precision. This ensures the tool reaches the desired areas of the sculpture.
  2. **Sensing Forces:** The robot arm has sensors that detect the force applied by the tool on the sculpture.
  3. **Adapting to Pressure:** If the force gets too high (indicating excessive pressure), the system slightly adjusts the robot's position to maintain a gentle touch. This prevents accidental gouges or breaks in the sculpture.
  4. **Maintaining Precision:** Despite these adjustments, the overall positioning remains controlled, allowing you to sculpt delicate details.

Benefits of Hybrid Control:

- **Versatility:** Enables robots to perform tasks requiring both precise positioning and delicate force control.
- **Improved Accuracy:** Reduces the risk of damaging objects while maintaining precise control over the robot's movements.
- **Wider Applications:** Allows robots to handle tasks like assembly, polishing, or surgical procedures where both precise manipulation and sensitivity to force are crucial.

★★★

Reference Books:

1. Dilip Kumar Pratihar, Fundamentals of Robotics, Narosa Publishing House 2019.
2. S. K. Saha, Introduction to robotics 2e, Tata MacGraw Hills Education 2014.
3. J. J. Craig, Introduction to Robotics: Mechanics & Control, Addison Wesley 2003.

NPTEL video lectures reference:

1. https://archive.nptel.ac.in/noc/courses/noc20/SEM2/noc20-me56