Advanced Robotics Note

Authors: Haobo Yang (GitHub: https://github.com/YangSierCode000)
Source: https://github.com/YangSierCode000/aroNote (welcoming pull request)
Reference: The University of Edinburgh, Advanced Robotics, course link
Update date: 19/12/2023

1. Geometry

1.1. 3D State Description in Robotics

1.1.1. Demo Case Study: 3D Navigation of a Drone in an Urban Environment

- **Super Domain**: Path & Motion Planning
- **Type of Method**: State Representation in 3D Space

1.1.2. Problem Definition and Variables

- **Objective**: To represent and track the state of a drone for 3D navigation in an urban environment.
- **Variables**:
  - q: State vector including position and orientation in 3D space.
  - (x, y, z): 3D Cartesian coordinates for position.
  - (ϕ, θ, ψ): Roll, pitch, and yaw angles for orientation.
  - Environmental variables: Obstacles' positions, wind conditions, goal location.

1.1.3. Assumptions

- The drone operates in a three-dimensional space.
- External conditions like wind and air traffic are either known or can be sensed.

1.1.4. Method Specification and Workflow

1. **State Representation**: Define $q = [x, y, z, \phi, \theta, \psi]^T$.
2. **3D Environment Mapping**: Utilize 3D maps or LIDAR data for environmental representation.
3. **Sensor Integration**: Use GPS, gyroscopes, accelerometers, and LIDAR for real-time state updates.

4. **Advanced State Estimation**: Implement algorithms like Extended Kalman Filter or SLAM (Simultaneous Localization and Mapping) for accurate state estimation in 3D.

1.1.5. Strengths and Limitations

- **Strengths**:
  - Comprehensive representation of the drone's position and orientation in 3D space.
  - Facilitates complex navigation tasks, including obstacle avoidance and route optimization.
  - Adaptable to a variety of 3D navigation tasks in different environments.
- **Limitations**:
  - High computational requirements for real-time state estimation and mapping.
  - Sensor accuracy and reliability can significantly affect state estimation.
  - Complex environmental dynamics can challenge state representation accuracy.

1.1.6. Common Problems in Application

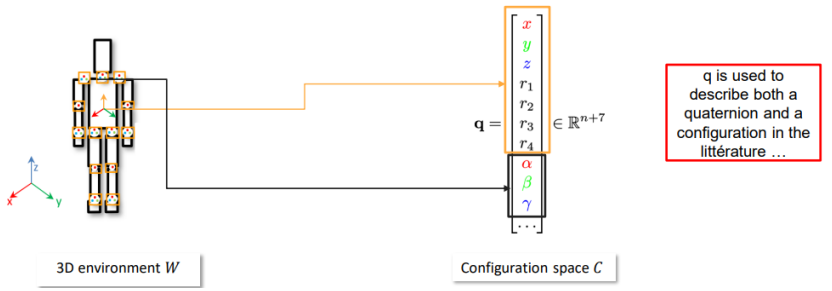
- Drift in state estimation due to sensor errors, especially in GPS-denied environments.
- Difficulty in mapping and navigating in cluttered or dynamically changing environments.
- Balancing computational load and real-time performance requirements.

1.1.7. Improvement Recommendations

- Integrate more robust and diverse sensors (e.g., visual odometry) to enhance state estimation.
- Employ machine learning techniques for predictive modeling of environmental dynamics.
- Optimize algorithms for efficient computation, considering the constraints of onboard processing capabilities.

The configuration space (Lozano-Peréz 83)

- Robot posture is a point **q** in the configuration space C, of dimension n, or n+6 if root is free (free-flyer joint), with n number of internal Degrees Of Freedom (dof)
  - Each internal dof represented by a joint parameter, subset of **q**
  - If using quaternions to represent free-flyer rotation, **q** is ***represented*** with n+7 != n+6 variables



1.2. Euler Angles and Gimbal Lock in Robotics

1.2.1. Demo Case Study: Orientation Control of a Robotic Camera System

- **Super Domain**: Kinematics and Control Systems
- **Type of Method**: Orientation Representation and Control

1.2.2. Problem Definition and Variables

- **Objective**: To control the orientation of a robotic camera system using Euler angles while understanding and mitigating the risks of gimbal lock.
- **Variables**:
  - Euler Angles (ϕ, θ, ψ): Roll, pitch, and yaw angles representing the orientation of the camera.
  - Gimbal Lock: A condition where two of the three gimbals align, resulting in a loss of one degree of rotational freedom.

1.2.3. Assumptions

- The camera system operates under a gimbal mechanism allowing three degrees of rotational freedom.
- The motion and orientation of the system can be accurately measured and controlled using Euler angles.

1.2.4. Method Specification and Workflow

1. **Euler Angle Representation**: Define orientation using three rotations about principal axes.
   - Sequence of rotations (e.g., Z-Y-X) is crucial for defining the orientation.
2. **Gimbal Lock Understanding**: Recognize that gimbal lock occurs when the pitch angle (θ) is at 90°, aligning two of the three rotation axes.
3. **Orientation Control**: Implement control algorithms to adjust roll, pitch, and yaw for desired camera orientation.
4. **Gimbal Lock Mitigation**: Monitor and manage the pitch angle to avoid approaching critical values where gimbal lock can occur.

1.2.5. Strengths and Limitations

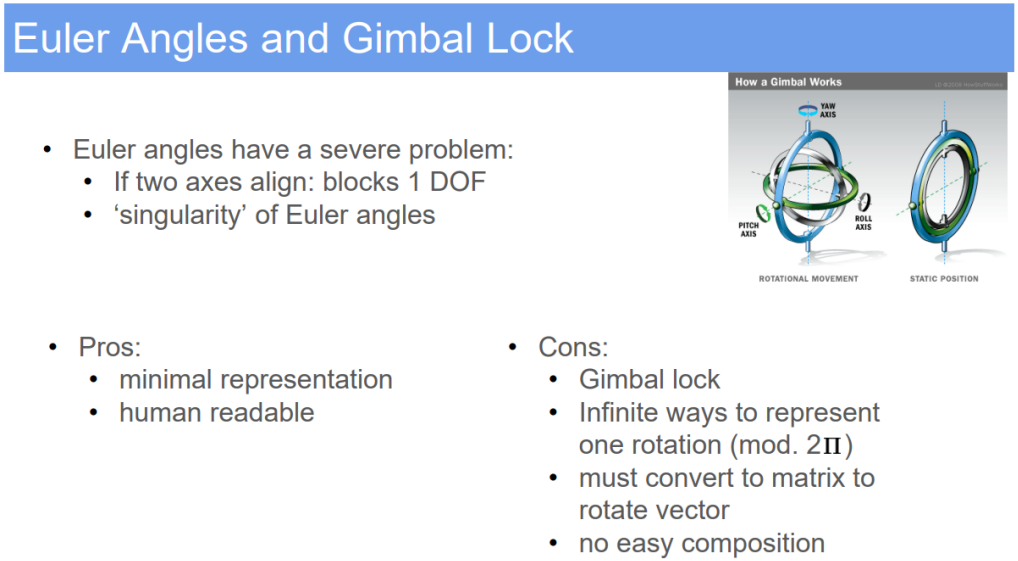
- **Strengths**:
  - Euler angles provide an intuitive way to describe orientation.
  - Suitable for applications with limited rotational movement where gimbal lock can be avoided.
- **Limitations**:
  - Gimbal lock leads to a loss of one rotational degree of freedom, making precise control impossible in certain orientations.
  - Euler angles can be less intuitive for complex, three-dimensional rotational movements.

1.2.6. Common Problems in Application

- Unintentionally entering gimbal lock, resulting in control issues.
- Difficulty in visualizing and programming complex rotations using Euler angles.
- Accumulation of numerical and computational errors, especially near gimbal lock configurations.

1.2.7. Improvement Recommendations

- Implement quaternion-based representation as an alternative to avoid gimbal lock.
- Use advanced control algorithms that can detect and compensate for approaching gimbal lock conditions.
- Educate operators and programmers about the limitations and risks of gimbal lock in systems using Euler angles.



1.3. 3D State Description in Robotics with Quaternion Improvement

1.3.1. Demo Case Study: Advanced 3D Navigation of an Aerial Robot in Complex Environments

- **Super Domain**: Path & Motion Planning
- **Type of Method**: Enhanced State Representation in 3D Space

1.3.2. Problem Definition and Variables

- **Objective**: To accurately represent and manage the 3D state of an aerial robot for complex navigation tasks.
- **Variables**:
  - q: Enhanced state vector including position and orientation in 3D space.
  - (x, y, z): 3D Cartesian coordinates for position.
  - Quaternion (q_w, q_x, q_y, q_z): Representing orientation to overcome Euler angle limitations.

1.3.3. Assumptions

- The robot operates in a dynamic, three-dimensional environment.
- High accuracy in orientation representation is crucial for navigation and task execution.

1.3.4. Method Specification and Workflow

1. **State Representation**: Define $q = [x, y, z, q_w, q_x, q_y, q_z]^T$.
2. **Quaternion Usage**: Replace Euler angles with quaternions to avoid issues like gimbal lock and provide more accurate and stable orientation representation.
3. **Sensor Fusion**: Integrate IMU data with visual odometry and LIDAR for robust state estimation.
4. **Advanced Algorithms**: Use state-of-the-art algorithms like Unscented Kalman Filter or SLAM for precise state tracking in 3D space.

1.3.5. Strengths and Limitations

- **Strengths**:
  - Quaternions provide a more robust and singularity-free representation of orientation in 3D space.
  - Enhanced precision in orientation helps in complex maneuvers and accurate navigation.
  - Suitable for high-dynamic environments like aerial robotics in urban or natural terrains.
- **Limitations**:
  - Increased complexity in understanding and implementing quaternion mathematics.
  - Higher computational demands for real-time quaternion-based state estimation.
  - Requires sophisticated sensor fusion techniques to fully leverage quaternion advantages.

1.3.6. Common Problems in Application

- Complexity in translating quaternion operations into practical control commands.
- Challenges in seamlessly integrating quaternion data with traditional position sensors.
- Ensuring numerical stability and avoiding normalization issues with quaternions.

1.3.7. Improvement Recommendations

- Integrate machine learning to predict and compensate for environmental dynamics and sensor inaccuracies.
- Develop specialized algorithms for efficient quaternion normalization and operation.
- Utilize hardware acceleration for real-time processing of complex quaternion-based calculations.

*Rotation: Quarternion

A quaternion is $r \in \mathbb{R}^4$

$$\mathbf{r} = \begin{bmatrix} sin(\theta/2)u_1 \\ sin(\theta/2)u_2 \\ sin(\theta/2)u_3 \\ cos(\theta/2) \end{bmatrix}$$

with **u** = [u_1, u_2, u_3] unit rotation axis

Unit length constraint (to represent rotations)

$$r^\top r = r_0^2 + r_1^2 + r_2^2 + r_3^2 = 1$$

A single rotation can be defined by two different quaternions (this will matter!)

Exercise:
What is the equivalent quaternion?

A map can be obtained to apply a rotation r to p

$$\mathbf{r_{p'}} = \mathbf{r r_p \bar{r}}$$

With r_p= [p_x, p_y, p_z, 0] (pure quaternion)

1.4. Special groups for rotations

Special groups for rotations

An element of the group….		Can be represented as …	
$r \in SO(3)$ rotation	$\simeq$	$\mathbf{R} \in \mathbb{R}^{3\times3}$	Rotation matrix
		$\mathbf{q} \in \mathbb{H} \simeq \mathbb{R}^4, \|\mathbf{q}\| = 1$	quaternion
		$\mathbf{w} \in so(3) \simeq \mathbb{R}^3$	
$m \in SE(3)$ displacement	$\simeq \mathbb{R}^3 \times SO(3)$ (translation, rotation) $\simeq$	$\mathbb{R}^3 \times \mathbb{R}^{3\times3} \simeq \mathbb{R}^{4\times4}$	Homogeneous matrix
		$\mathbb{R}^3 \times \mathbb{H} \simeq \mathbb{R}^7$	
		$\mathcal{V} \in se(3) = \begin{bmatrix} \mathbf{v} \\ \mathbf{w} \end{bmatrix} \simeq \mathbb{R}^6$	

1.5. Displacement in Robotics (Translation and Rotation)

1.5.1. Demo Case Study: Precise Manipulator Movement in Industrial Robotics

- **Super Domain**: Kinematics
- **Type of Method**: Geometric and Kinematic Analysis

1.5.2. Problem Definition and Variables

- **Objective**: To accurately describe and control the displacement of a robotic manipulator, including both translation and rotation.
- **Variables**:
  - $\mathbf{p}$: Position vector representing translation in Cartesian coordinates (x, y, z).
  - Rotation: Represented by Euler angles (ϕ, θ, ψ), rotation matrices, or quaternions.

1.5.3. Assumptions

- The manipulator operates within a rigid body framework.
- Joint movements are precise and can be accurately measured.

1.5.4. Method Specification and Workflow

1. **Translation**: Described by linear displacement along x, y, and z axes.
   - Formula: $\Delta\mathbf{p} = \mathbf{p}_{final} - \mathbf{p}_{initial}$.
2. **Rotation**:
   - Euler Angles: Changes in orientation described by roll (ϕ), pitch (θ), and yaw (ψ).
   - Rotation Matrix: $\mathbf{R} = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi)$.
   - Quaternion: $q = [q_w, q_x, q_y, q_z]$, providing a singularity-free representation.
3. **Homogeneous Transformation Matrix**: Combines translation and rotation in a single matrix.
   - Formula: $\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix}$.

1.5.5. Strengths and Limitations

- **Strengths**:
  - Provides a comprehensive way to describe both position and orientation.
  - Suitable for a wide range of robotic applications, including automation and assembly.
  - Facilitates the calculation of kinematic chains and motion planning.
- **Limitations**:
  - Complexity in calculations, especially for rotations in 3D space.
  - Euler angles can suffer from gimbal lock in certain configurations.
  - Precision depends on the accuracy of the measurement systems.

1.5.6. Common Problems in Application

- Accumulation of numerical errors in successive transformations.
- Complexity in inverse kinematics for determining joint angles from desired end-effector position.
- Difficulty in intuitively understanding and visualizing quaternion-based rotations.

1.5.7. Improvement Recommendations

- Implement advanced computational tools for handling complex kinematic calculations.
- Utilize quaternions for rotation to avoid gimbal lock and improve computational efficiency.
- Integrate feedback systems for error correction in real-time motion control.

Homogeneous Transformation Matrix trick

❑ ${}^A\mathbf{p} = {}^A\mathbf{R}_B {}^B\mathbf{p} + \mathbf{t}$ => annoying to write (especially when composing)

❑ This operation can be written in matrix form:

$$\begin{bmatrix} {}^A\mathbf{p} \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} {}^A\mathbf{R}_B & t \\ \mathbf{0}_3 & 1 \end{bmatrix}}_{{}^A\mathbf{M}_B \in \mathbb{R}^{4\times4}} \begin{bmatrix} {}^B\mathbf{p} \\ 1 \end{bmatrix}$$

❑ With

$${}^B\mathbf{M}_A = ({}^A\mathbf{M}_B)^{-1} = \begin{bmatrix} {}^B\mathbf{R}_A & -{}^B\mathbf{R}_A t \\ \mathbf{0}_3 & 1 \end{bmatrix}$$

2. Kinematics

2.1. Kinematic Tree, Kinematic Chain, and Forward Geometry and Backward Geometry in Robotics

2.1.1. Demo Case Study: Robotic Arm for Assembly Line Tasks

- **Super Domain**: Kinematics in Robotics
- **Type of Method**: Kinematic Analysis and Control

2.1.2. Kinematic Tree and Kinematic Chain

- **Objective**: To model the structure and movement of a robotic arm.
- **Kinematic Tree**:
  - Represents the hierarchical structure of a robot.
  - Includes all possible movements and connections between different components.
- **Kinematic Chain**:
  - A sequence of links and joints from the base to the end effector.
  - Typically, a subset of the full kinematic tree, focusing on a specific task.

2.1.3. Forward Geometry

- **Objective**: To determine the position and orientation of the robot's end effector, given its joint parameters.
- **Variables**:
  - $\mathbf{q}$: Vector of joint parameters (angles for revolute joints, displacements for prismatic joints).
  - $\mathbf{T}$: Homogeneous transformation matrix representing end effector position and orientation.
- **Method**:
  - Utilize the kinematic chain to calculate $\mathbf{T}$ as a function of $\mathbf{q}$.

2.1.4. Inverse Geometry

- **Objective**: To compute the joint parameters that achieve a desired position and orientation of the end effector.
- **Method**:
  - More complex than forward kinematics, often requiring numerical methods (find inverse of forward kinematics) or iterative algorithms (check inverse kinematics).

2.1.5. Jacobian Matrix

- **Objective**: To relate the velocities in the joint space to velocities in the Cartesian space.
- **Variables**:
  - $\mathbf{J}$: Jacobian matrix.
- **Method**:
  - Construct $\mathbf{J}$ such that $\dot{\mathbf{x}} = \mathbf{J}\dot{\mathbf{q}}$, where $\dot{\mathbf{x}}$ is the velocity in Cartesian space and $\dot{\mathbf{q}}$ is the velocity in joint space.

2.1.6. Strengths and Limitations

- **Strengths**:
  - Provides a systematic approach to model and control robotic mechanisms.
  - Facilitates the understanding of complex robotic systems and their motion capabilities.
  - Essential for the design and control of robotic systems in various applications.
- **Limitations**:
  - Inverse kinematics can be computationally intensive and may not have a unique solution.
  - Jacobian matrices can become singular, leading to control difficulties.

2.1.7. Common Problems in Application

- Singularities in the kinematic chain, leading to loss of control or undefined behavior.
- Accuracy issues in forward and inverse kinematics due to mechanical tolerances and sensor errors.
- Computational complexity, especially for robots with a high degree of freedom.

2.1.8. Improvement Recommendations

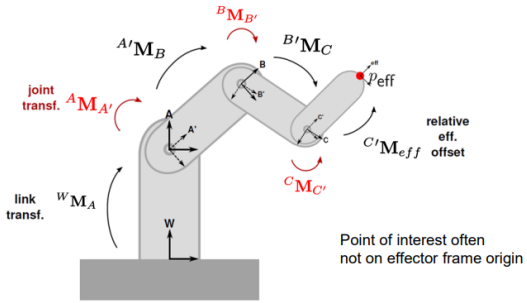
- Implement redundancy management strategies to handle singularities.
- Use advanced algorithms like pseudo-inverse or optimization-based methods for inverse kinematics.
- Integrate real-time feedback and adaptive control mechanisms to compensate for inaccuracies.

Kinematic chain and map

Generally, frames placed as follows to simplify the variable transformations

Black placements are constant

Red placement are functions of q



$$^{W}\mathbf{M}_{eff}(\mathbf{q}) = {}^{W}\mathbf{M}_{A}\,{}^{A}\mathbf{M}_{A'}(\mathbf{q})\,{}^{A'}\mathbf{M}_{B}\,{}^{B}\mathbf{M}_{B'}(\mathbf{q})\,{}^{B'}\mathbf{M}_{C}\,{}^{C}\mathbf{M}_{C'}(\mathbf{q})\,{}^{C'}\mathbf{M}_{eff}$$

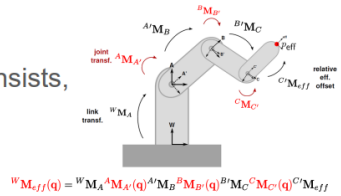
Forward and inverse geometry

❑ The computation of any map f(q) relates to **forward geometry**

FG: q ⟶ x = f(q) , x in SE(3)

❑ The inverse problem, called **inverse geometry**, consists, given a desired x*, in finding a q such that f(q) = x*

IG: x* ⟶ q = f⁻¹(x*)



In the interesting cases, f is almost never invertible => numerical approaches through optimisation:

Search **q** such that f(q) = x* <=> min distance(f(q), x*) => end of tutorial1

2.2. Spatial Velocity Vector in Robotics

2.2.1. Demo Case Study: Calculating the Movement of a Robotic Arm's End Effector

- **Super Domain**: Kinematics and Dynamics
- **Type of Method**: Vector Analysis

2.2.2. Problem Definition and Variables

- **Objective**: To represent and compute the motion of a robotic arm's end effector through its spatial velocity, which includes both linear and angular components.
- **Variables**:
  - $\mathbf{v}$: Linear velocity component of the spatial velocity vector.
  - ω: Angular velocity component of the spatial velocity vector.
  - ${}^{A}\mathbf{v}_{BC}$: Spatial velocity of frame C with respect to frame B, expressed in frame A.

2.2.3. Assumptions

- The frames of reference are rigid, with no deformation under motion.
- The motion can be captured within the Euclidean space $se(3)$ framework.

2.2.4. Method Specification and Workflow

1. **Spatial Velocity Representation**: Use the $se(3)$ group to describe the spatial velocity as a six-dimensional vector combining linear and angular velocity.
   - Representation: $\mathbf{v} \in se(3) = \begin{bmatrix} \mathbf{v}^A \\ \omega^A \end{bmatrix}$.
2. **Velocity Composition**: Determine the overall spatial velocity of the end effector (frame C) with respect to the base (frame A) by summing up individual velocities along the kinematic chain.
   - Composition formulas:

$$^{A}\mathbf{v}_{AC} = {}^{A}\mathbf{v}_{AB} + {}^{A}\mathbf{v}_{BC}$$
$$^{A}\mathbf{v}_{AC} = {}^{A}\mathbf{v}_{AB} + {}^{A}X_{B}\,{}^{B}\mathbf{v}_{BC}$$

3. **Transformation Matrix Utilization**: Apply the appropriate transformation matrices to express all velocities in a common frame of reference, typically the base frame.

2.2.5. Strengths and Limitations

- **Strengths**:
  - Provides a complete description of motion, essential for control and analysis.
  - Can be used to compute velocities for any point along a robotic arm or system.
- **Limitations**:
  - Requires precise knowledge of the kinematic chain and transformations.
  - The computation can become complex, especially for systems with multiple moving parts.

2.2.6. Common Problems in Application

- Errors in calculating transformation matrices can lead to incorrect velocity composition.
- The intricacy of multi-body systems can introduce challenges in real-time applications.

2.2.7. Improvement Recommendations

- Implement software frameworks that can automatically handle spatial transformations and velocity computations.

- Use state estimation techniques to accurately measure and update velocities, reducing errors from model inaccuracies.
- Integrate sensor feedback for real-time correction of spatial velocities.

Spatial velocity vector

❑ **v** linear velocity

❑ ω angular velocity

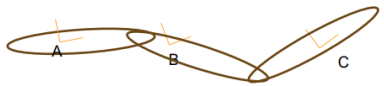
$$^{A}\mathcal{V} \in se(3) = \begin{bmatrix} ^{A}\mathbf{v}_{A} \\ ^{A}\omega \end{bmatrix} \simeq \mathbb{R}^6$$

❑ $^{A}\mathcal{V}_{BC}$: spatial velocity of frame C wrt to frame B, expressed in frame A

$$^{A}\mathcal{V}_{AC} = ^{A}\mathcal{V}_{AB} + ^{A}\mathcal{V}_{BC}$$

$$^{A}\mathcal{V}_{AC} = ^{A}\mathcal{V}_{AB} + ^{A}\mathbf{X}_{B}\,^{B}\mathcal{V}_{BC}$$

with $^{A}X_{B}$ the « adjoint » or « action » matrix



$$^{A}\mathbf{X}_{B} = \begin{bmatrix} ^{A}\mathbf{R}_{B} & ^{A}\overrightarrow{\mathbf{AB}}_{\times}\,^{A}\mathbf{R}_{B} \\ 0 & ^{A}\mathbf{R}_{B} \end{bmatrix}$$

Let's just accept this for now

2.3. Forward Kinematics for a Chain of Joints

2.3.1. Demo Case Study: Movement Analysis of a Robotic Arm

- **Super Domain**: Kinematics in Robotics
- **Type of Method**: Analytical Modeling

2.3.2. Problem Definition and Variables

- **Objective**: To determine the position and orientation of a robotic arm's end effector based on joint configurations.
- **Variables**:
  - q: Vector of joint parameters (e.g., angles for revolute joints, displacements for prismatic joints).
  - $J(q)$: Jacobian matrix at configuration q.
  - $v(q, \dot{q})$: Velocity of the end effector in Cartesian space.
  - $\dot{q}$: Vector of joint velocities.

2.3.3. Assumptions

- The robotic arm is composed of a series of rigid bodies connected by joints, forming a kinematic chain.
- The kinematic model of the robot is known and can be described by a set of parameters q.

2.3.4. Method Specification and Workflow

1. **Jacobian Computation**: Calculate the Jacobian matrix $J(q)$, which relates joint velocities to end effector velocities in Cartesian space.
2. **Velocity Mapping**: Use the Jacobian to map joint velocities $\dot{q}$ to the velocity of the end effector v in Cartesian space using the formula:
   - $v(q, \dot{q}) = J(q)\dot{q}$.
3. **Local Validity**: Understand that the Jacobian is only valid locally, meaning it accurately describes the relationship between joint velocities and end effector velocities only in the vicinity of the configuration q.

2.3.5. Strengths and Limitations

- **Strengths**:
  - Provides a direct and efficient way to calculate the end effector's position and orientation.
  - Essential for real-time control and path planning of robotic manipulators.
- **Limitations**:
  - The Jacobian does not provide global information about the robot's workspace.
  - Singularities in the Jacobian matrix can lead to issues in control algorithms.

2.3.6. Common Problems in Application

- Computational complexity for robots with a high degree of freedom.
- Inaccuracies in the Jacobian matrix due to modeling errors or physical changes in the robot's structure.

2.3.7. Improvement Recommendations

- Implement real-time Jacobian updates to accommodate changes in the robot's configuration and avoid singularities.
- Utilize redundancy in robotic design to handle singularities and increase the robustness of the kinematic model.
- Integrate advanced sensors and feedback systems to refine the accuracy of the Jacobian matrix continually.

Forward kinematics for a chain of joints:

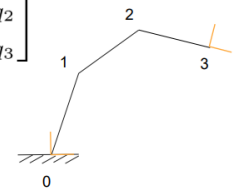
$$^0\nu_{03}(q, v_q) = {}^0\nu_{01} + {}^0\nu_{12} + {}^0\nu_{23}$$

$$^0\nu_{03}(q, v_q) = {}^0\mathbf{X}_1 {}^1\nu_{01} + {}^0\mathbf{X}_2 {}^2\nu_{12} + {}^0\mathbf{X}_3 {}^3\nu_{23}$$

$$^0\nu_{03}(q, v_q) = {}^0\mathbf{X}_1 {}^1J_1(q_1)v_{q_1} + {}^0\mathbf{X}_2 {}^2J_2(q_2)v_{q_2} + {}^0\mathbf{X}_3 {}^3J_3(q_3)v_{q_3}$$

$$^0\nu_{03}(q, v_q) = \begin{bmatrix} {}^0\mathbf{X}_1 {}^1J_1(q_1) & {}^0\mathbf{X}_2 {}^2J_2(q_2) & {}^0\mathbf{X}_3 {}^3J_3(q_3) \end{bmatrix} \begin{bmatrix} v_{q_1} \\ v_{q_2} \\ v_{q_3} \end{bmatrix}$$

$$\boxed{\nu(\mathbf{q}, v_\mathbf{q}) = \mathbf{J}(\mathbf{q})v_\mathbf{q}}$$



2.4. Inverse Kinematics (IK) and Inverse Geometry (IG) in Robotics

2.4.1. Demo Case Study: Configuring a Robotic Arm to Reach a Target Position

- **Super Domain**: Kinematics and Numerical Optimization
- **Type of Method**: Mathematical Problem-Solving

2.4.2. Problem Definition and Variables

- **Objective**: To determine the joint parameters that will position a robotic arm's end effector at a desired location and orientation.
- **Variables**:
  - IK: Inverse Kinematics, focusing on finding joint velocities $\dot{q}$ to achieve a desired end-effector velocity ν^*.
  - IG: Inverse Geometry, solving for joint positions q that reach a specific end-effector pose.
  - $J(q)$: Jacobian matrix at joint configuration q, relating joint velocities to end-effector velocities.

2.4.3. Assumptions

- The kinematic structure of the robotic arm is well-defined and the Jacobian matrix can be computed.

2.4.4. Method Specification and Workflow

1. **IK Formulation**:
   - Solve $\min_{\dot{q}} \|\nu(q, \dot{q}) - \nu^*\|^2$, where $\nu(q, \dot{q}) = J(q)\dot{q}$.
   - Utilize the Moore-Penrose pseudo-inverse J^+ when the Jacobian is not invertible.
2. **IG Formulation**:
   - Solve $\min_q \text{dist}({}^0M_e(q), {}^0M_*)$ for pose and $\min_{\dot{q}} \text{dist}({}^0M_e(q_0 \oplus \dot{q}), {}^0M_*)$ for velocity, with ${}^0M_e$ denoting the end-effector pose in the base frame and ${}^0M_*$ being the target pose.
3. **IK with Constraints**:
   - Consider velocity bounds $\dot{q}^- \leq \dot{q} \leq \dot{q}^+$ and joint limits $q^- \leq q + \Delta t\dot{q} \leq q^+$ to the optimization problem.
   - Solve using a Quadratic Program (QP) solver like 'quadprog'.

2.4.5. Strengths and Limitations

- **Strengths**:
  - IK is a linear problem and generally easier to solve, providing a precise control mechanism for robotic motion.
  - IG offers a more direct approach to reaching a specific target pose.
- **Limitations**:
  - The Jacobian matrix may not always be invertible, necessitating pseudo-inverse solutions.
  - IG is a nonlinear problem and can be computationally challenging, often requiring iterative methods.

2.4.6. Common Problems in Application

- Jacobian singularities can lead to control problems and undefined velocities.
- High computational cost for iterative solutions in IG.
- Difficulty in ensuring convergence to the global minimum for IG problems.

2.4.7. Improvement Recommendations

- Apply redundancy in the robotic design to mitigate the effects of singularities.
- Use optimization libraries that are robust to numerical issues and can handle constraints effectively.
- For IG, consider gradient descent methods and initialization strategies to improve convergence.

IK vs IG

- ❑ Inverse kinematics (also called differential IK) is a linear, convex problem, very easy to solve
- ❑ Inverse geometry (also called IK) is a non-linear problem, very hard to solve
- ❑ When trying to solve IG iteratively, we can use the pseudo-inverse of the jacobian to locally update a configuration towards one that is closer to the goal. This is similar to performing one step of gradient descent (See example after)

3. Motion planning

3.1. Motion Planning in Robotics

3.1.1. Demo Case Study: Navigating an Autonomous Rover on Mars

- **Super Domain**: Motion Planning
- **Type of Method**: Pathfinding in Complex Environments

3.1.2. Problem Definition and Variables

- **Objective**: Plan collision-free motions for a rover from a start to a goal position among a set of obstacles on a Martian landscape.
- **Variables**:
  - Environmental model, including terrain map and obstacle positions.
  - Rover's capabilities, including its movement constraints and geometry.

3.1.3. Method Specification and Workflow

- **Motion Planning**: Develop a strategy to compute a feasible path that the rover can follow to reach the goal without collisions.
- **Path Planning Problem**: Define a path planning problem with specific start and goal points, considering the rover's and environment's properties.
- **Approaches**:
  - **Local Methods**: Reactively adapt the behavior of the rover based on immediate sensor data to avoid obstacles.
  - **Global Methods**: Precompute a path considering the entire map, designed to avoid local minima and ensure reaching the goal.

3.1.4. Strengths and Limitations

- **Strengths**:
  - Allows for the safe navigation of robots in unknown or dynamic environments.
  - Can be adapted for different types of robots and objectives.
- **Limitations**:
  - The complexity of the environment can make the problem computationally intensive.
  - Real-time changes in the environment may require re-planning.

3.1.5. Common Problems in Application

- The rover may encounter unexpected obstacles not present in the initial map.
- Varying terrain properties may affect the rover's ability to follow a planned path.

3.1.6. Improvement Recommendations

- Integrate sensory feedback into planning algorithms for dynamic adaptation.
- Employ machine learning techniques to predict and plan for environmental changes.

Motion Planning

Fundamental task: plan collision-free motions for complex systems from a start to a goal position among a set of obstacles. For mobile robots, it is also referred as a path planning problem.

- ❏ **An intuitive approach: Potential Fields**
- ❏ Sampling based motion planning
  - ❏ Rapidly exploring Random Tree (RRT): principle and code demo (in tutorial)
    Variants of RRT algorithms
  - ❏ Multi-Query Planner: Probabilistic Roadmap (PRM)
- ❏ Grid-based search algorithms: A*

Sampling based motion planning summary

- ❏ We have (hopefully) come up with the principles for a global planning algorithm
- ❏ A sampling based motion planning algorithm generates a graph were:
  - ❏ Nodes are points in the feasible space (in our case C_{free})
  - ❏ Edges are feasible paths between Nodes computing with a local steering method:
    - ❏ In geometric case, often obtained by interpolation
    - ❏ Can be as complex as required by the considered problem
- ❏ The formulation is very generic and can be used to represent any robotics planning problem (RRTs were developed for vehicle control, ie differential constraints)

3.2. Potential Field in Robotics

3.2.1. Demo Case Study: Autonomous Robot Navigation in a Dynamic Environment

- **Super Domain**: Motion Planning
- **Type of Method**: Vector Field Approach

3.2.2. Problem Definition and Variables

- **Objective**: To navigate an autonomous robot through a dynamic environment to a goal position while avoiding obstacles.
- **Variables**:
  - U: The potential field in the environment.
  - $\nabla U(q)$: The gradient of the potential field, which indicates the direction of the force exerted by the field.

3.2.3. Method Specification and Workflow

- **Attractive Component**: Pulls the robot towards the goal, modeled by a potential well where the goal is the lowest point.
- **Repulsive Component**: Pushes the robot away from obstacles, modeled by potential peaks around the obstacles.
- **Principle**: Use a potential field that exerts virtual forces on the robot: attractive forces pull it towards the goal, while repulsive forces push it away from obstacles.
- **Workflow**:
  i. Define the attractive potential field around the goal position.
  ii. Define the repulsive potential field around the obstacles.
  iii. Combine the attractive and repulsive fields to create a composite potential field.
  iv. Navigate the robot in the direction of the negative gradient of the composite field.

3.2.4. Strengths and Limitations

- **Strengths**:
  - Computationally fast and capable of producing smooth paths.
  - Intuitive and easy to implement.
  - Produces natural motions if configured correctly.
- **Limitations**:
  - Can get trapped in local minima, failing to find a path to the goal.
  - May produce unstable oscillations in the presence of obstacles or narrow passages.
  - The solution is not guaranteed and may require manual tuning.
  - The method is not complete, meaning it does not guarantee that a solution will be found even if one exists, nor is it optimal.

3.2.5. Common Problems in Application

- The robot may oscillate indefinitely in narrow corridors due to equal repulsive forces from the walls.
- The robot may not find a path to the goal if the potential field has local minima away from the goal.

3.2.6. Improvement Recommendations

- Incorporate random walks or other exploration strategies to escape local minima.
- Use hybrid methods combining potential fields with other planning strategies for robustness.

3.2.7. Potential Field (PF) Discussion

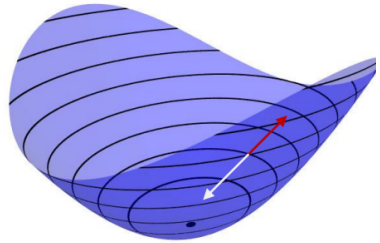
- Advanced techniques like the navigation function approach can mitigate some limitations of potential fields by ensuring convergence to the goal without getting trapped in local minima.
- The repulsive potential typically decreases with distance from the obstacle, becoming zero beyond a certain threshold.

Potential Fields

❑ Artificial potential field approach is originally proposed for collision avoidance. It constructs a differentiable real-valued function

$U : R^m \rightarrow R$

called a potential function, which guides the motion of the moving object.



❑Treat the value as 'energy'

❑Then, gradient is the vector,

$$\nabla U(q) = DU(q)' = [\frac{\partial U}{\partial q_1}(q), \ldots, \frac{\partial U}{\partial q_m}(q)]'$$

14

3.3. Rapidly-exploring Random Trees (RRT)

3.3.1. Demo Case Study: Autonomous Exploration Rover on Mars

- **Super Domain**: Motion Planning
- **Type of Method**: Sampling-based, Randomized Path Planning

3.3.2. Problem Definition and Variables

- **Objective**: Efficiently explore uncharted terrain on Mars to collect data, using a path planning approach that can quickly adapt to large, unknown areas.
- **Variables**:
  - Configuration space representing the rover's possible positions and orientations.
  - Current known configuration of the rover.

3.3.3. Method Specification and Workflow

- **Principle**: RRT is a data structure that represents a space-filling tree built from randomly sampled points in the configuration space. The tree rapidly expands towards unexplored areas, providing coverage and a path to the goal.
- **Workflow**:

i. Start with an initial node from the known configuration.
ii. Randomly sample a new point in the configuration space.
iii. Find the nearest node in the tree to the new sample.
iv. Extend the tree towards the new point by a predefined step size.
v. Check if the new extension is within the unexplored space and free of obstacles.
vi. Repeat the process until a path from the start to the goal configuration is identified.

3.3.4. Strengths and Limitations

- **Strengths**:
  - Rapidly explores vast areas, making it suitable for large-scale and high-dimensional spaces.
  - Doesn't require a pre-existing map of the environment.
- **Limitations**:
  - May not find the most optimal path due to the randomness of point sampling.
  - The basic RRT doesn't handle dynamic environments or changing obstacle configurations well.

3.3.5. Improvement Recommendations

- Implement RRT* or other variants which include a reconnection strategy to improve path optimality.
- Integrate with other sensors and algorithms for real-time re-planning in dynamic environments.

3.3.6. Rapidly-exploring Random Trees (RRT): Discussion

RRTs are particularly useful for single-query searches, where the objective is to find one viable path from a known configuration to a target as quickly as possible. This makes RRTs highly applicable to scenarios like space exploration or search and rescue missions, where time is of the essence and the environment may not be fully known in advance.

While the method is biased towards unexplored spaces, ensuring a comprehensive coverage over time, the random nature of the sampling process means that the quality of the path can vary. In mission-critical applications, this variability requires that the generated path be checked for safety and, if necessary, post-processed for smoothness and feasibility.

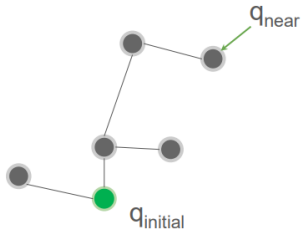
The adaptability of RRT to various mission parameters and environmental conditions, along with its relatively simple implementation, makes it a valuable tool in the field of autonomous system navigation. However, in complex environments with numerous obstacles, RRT may require modifications or combinations with other planning strategies to ensure robust performance.

Basic RRT algorithm

Pseudo code

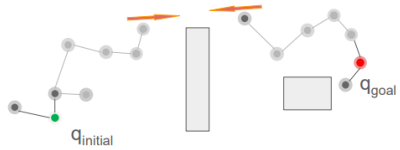
Algorithm 1 BUILD_RRT(q_{init})

$\mathcal{T}$.init(q_{init});
for $k = 1$ **to** K **do**
  $q_{rand} \leftarrow$ RANDOM_CONFIG();
  $q_{near} \leftarrow$ NEAREST_NEIGHBOR($q_{rand}, \mathcal{T}$);
  if edge_valid(q_{rand}, q_{near}) **then**
    $\mathcal{T}$.add_vertex(q_{rand});
    $\mathcal{T}$.add_edge(q_{near}, q_{rand});
return $\mathcal{T}$



RRT-Connect: grow and connect two RRTs

❑ A simple greedy heuristic that aggressively tries to connect *two trees*, one from the initial configuration and the other from the goal.



❑ The idea of constructing search trees from the initial and goal configurations comes from classical AI bi-directional search.

3.4. Probabilistic Roadmaps (PRM)

3.4.1. Demo Case Study: Path Planning for a Service Robot in a Hospital

- **Super Domain**: Motion Planning
- **Type of Method**: Multi-query, Sampling-based Planning

3.4.2. Problem Definition and Variables

- **Objective**: To enable a service robot to navigate efficiently through the complex corridors of a hospital, connecting various points of interest such as patient rooms, diagnostic centers, and pharmacies.
- **Variables**:
  - Configuration space representing the robot's possible positions and orientations.

- A set of points of interest that the robot may be required to visit multiple times.

3.4.3. Method Specification and Workflow

- **Principle**: PRM constructs a network or roadmap of possible paths by sampling the configuration space, adding valid samples as nodes, and connecting these nodes with edges if a direct path between them is possible and collision-free.
- **Workflow**:
  i. Randomly sample the configuration space to generate a set of nodes, discarding any that result in collisions with obstacles.
  ii. For each node, attempt to connect it to nearby nodes using a local planner to generate edges, creating the roadmap.
  iii. Once the roadmap is constructed, use graph search algorithms to find paths from any given start node to a goal node.
  iv. The roadmap can be reused for different start and goal configurations, making PRM suitable for multi-query scenarios.

3.4.4. Strengths and Limitations

- **Strengths**:
  - Well-suited for environments where the robot will need to perform multiple planning queries over time.
  - The precomputed roadmap allows for rapid path finding once constructed.
- **Limitations**:
  - Initial construction of the roadmap can be computationally intensive.
  - May not perform well in highly dynamic environments where obstacles frequently change positions.

3.4.5. Improvement Recommendations

- Optimize the sampling strategy to ensure coverage of critical areas while minimizing redundant nodes.
- Incorporate dynamic re-planning to update the roadmap in response to changes in the environment.

3.4.6. Probabilistic Roadmaps (PRM): Discussion

PRMs are a powerful tool for scenarios where a robot must navigate within a defined area and perform various tasks at different locations. By decoupling the roadmap construction from the path-finding phase, PRMs offer a flexible solution to the path planning problem, particularly in static or semi-static environments.

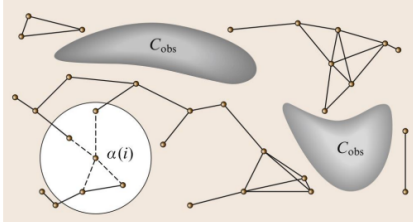
One of the key advantages of PRMs is the ability to handle complex, high-dimensional spaces, which is essential for robots with many degrees of freedom. However, PRMs are not without drawbacks. The quality of the roadmap is highly dependent on the sampling process, and there may be regions of the configuration space that are under-sampled, leading to potential gaps in connectivity.

Furthermore, the reliance on a static roadmap means that PRMs are less suited to environments with dynamic obstacles. In such cases, the roadmap may need frequent updates or a combination with local planning methods to ensure the robot can navigate safely and efficiently. Despite these challenges, PRMs remain one of the foundational techniques in robot motion planning, particularly in known and controlled spaces like hospital corridors, where the environment changes infrequently and predictably.

PRM: pseudo code

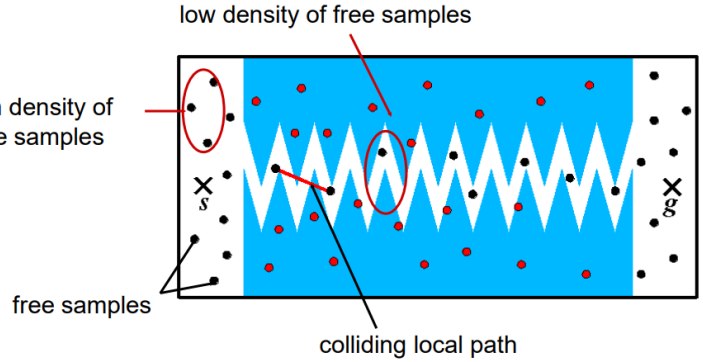
N: number of nodes to include in the roadmap

```
1:G.init(); i=0;
2:while i < N do
3:  if α(i) ∈ C_free then // is random config. collision-free
4:        G.add_ vertex(α(i)); i= i + 1;
5:        for q ∈ NEIGHBORHOOD(α(i),G) do
6:           if CONNECT (α(i),q) then
7:              G.add_ edge (α(i),q);
8:           end if
9:        end for
10:  end if
11:end while
```



α(i) is connected to all its neighbors.

What can Go Wrong? Narrow Passages



It is difficult to capture the free space associated with the narrow passages, because in the narrow passage, volume associated with the free space is relatively low.

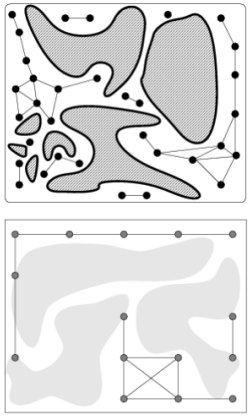
What Can Go Wrong? Lack of Connectivity

A combination of issues:

➢Collection of narrow passages can impede roadmap growth

➢Non-uniformity of a finite set of samples may lead to clustering – so k-nearest neighbor graph (k-NNG) connections may not cover the whole space

➢Similarly, boundary points of obstacles are sparsely sampled



4. Dynamics

Forward and inverse dynamics

❑ As for geometry and kinematics, we are interested in two formulations of the dynamics of a system

❑ Forward dynamics: Given $\mathbf{q}$, $\mathbf{v_q}$ and $\boldsymbol{\tau}$, compute joint accelerations $\dot{\mathbf{v}}_\mathbf{q}$

Useful for *simulation*

❑ Inverse dynamics: Given $\mathbf{q}$, $\mathbf{v_q}$ and $\dot{\mathbf{v}}_\mathbf{q}$, compute torque commands $\boldsymbol{\tau}$

Useful for *control*

4.1. Spring-Damper System Analysis

4.1.1. Demo Case Study: Vehicle Suspension System Design

- **Super Domain**: Mechanical Engineering and Dynamics
- **Type of Method**: Analytical Modeling

4.1.2. Problem Definition and Variables

- **Objective**: To design a vehicle suspension system that maximizes comfort by minimizing the impact of road irregularities on the vehicle's passengers.
- **Variables**:
  - $x(t)$: Displacement of the vehicle body or suspension component at time t.
  - m: Mass of the vehicle or the portion of the vehicle supported by the suspension.
  - k: Spring constant, representing the stiffness of the suspension springs.
  - c: Damping coefficient, representing the damping properties of the shock absorbers.
  - $F_{external}(t)$: External forces acting on the system, such as road bumps.

4.1.3. Method Specification and Workflow

- **System Dynamics**: The spring-damper system's behavior can be described by a second-order linear differential equation:
  - $m\ddot{x}(t) + c\dot{x}(t) + kx(t) = F_{external}(t)$
- **Analysis**:
  i. Derive the system's natural frequency and damping ratio to understand its free oscillation characteristics.
     - Natural Frequency (ω_n): $\omega_n = \sqrt{\frac{k}{m}}$
     - Damping Ratio (ζ): $\zeta = \frac{c}{2\sqrt{mk}}$
  ii. Determine the system's response to external forces, considering both transient and steady-state behavior.
  iii. Analyze the system's stability and performance under different loading conditions and external force profiles.

4.1.4. Strengths and Limitations

- **Strengths**:
  - The spring-damper model is well-understood and widely applicable to various mechanical systems.
  - Provides a straightforward means of designing for specific performance characteristics, like ride comfort or handling.
- **Limitations**:
  - Simplifications in the model may not capture all real-world behaviors, such as non-linearities in the spring or damping forces.
  - Assumes that all external forces are known and can be modeled, which may not be the case in unpredictable environments.

4.1.5. Common Problems in Application

- Non-linear behavior in the springs or damping elements can lead to deviations from the predicted response.
- Wear and environmental factors can change the system parameters over time, leading to performance degradation.

4.1.6. Improvement Recommendations

- Incorporate variable stiffness and damping elements to adapt to changing conditions and improve system robustness.
- Use feedback control systems, like semi-active or active suspensions, to dynamically adjust the system's response.

4.1.7. Discussion on Spring-Damper Systems

Spring-damper systems form the basis of many mechanical systems designed to absorb energy and mitigate the transmission of shock and vibration. In the context of vehicle suspensions, the interplay between the spring and damper is critical for ensuring that the vehicle can maintain contact with the road surface while providing a smooth ride.

Optimizing a spring-damper system involves a trade-off between stiffness for handling and softness for comfort. The correct balance depends on the intended use of the vehicle; for example, a sports car may have a stiffer setup for better handling, while a luxury car may prioritize a softer setup for comfort.

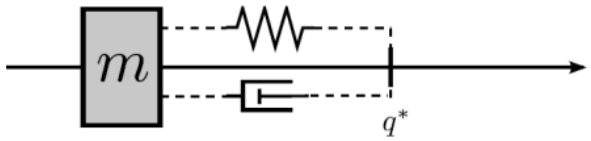
Advanced systems may include additional components like anti-roll bars for improved stability or pneumatic or hydraulic systems for adjustable ride height and stiffness. The design process is iterative and often relies on both computational models and real-world testing to fine-tune the system for the desired performance outcomes.

Spring-damper system

$$u = K_p(q^* - q) + K_d(\dot{q}^* - \dot{q}) + K_i \int_{s=0}^{t} (q^*(s) - q(s))\, ds$$

- **PID control**
  - Proportional Control ("Position Control")
    $f \propto K_p(q^* - q)$
  - Derivative Control ("Damping")
    $f \propto K_d(\dot{q}^* - \dot{q})$ $(\dot{x}^* = 0 \rightarrow \text{damping})$
  - Integral Control ("Steady State Error")
    $f \propto K_i \int_{s=0}^{t}(q^*(s) - q(s))\, ds$

25



4.2. Robotics Dynamics

4.2.1. Demo Case Study: Industrial Robotic Arm for Assembly Line

- **Super Domain**: Robotics and Mechanical Systems
- **Type of Method**: Physics-based Modeling

4.2.2. Problem Definition and Variables

- **Objective**: To accurately predict and control the motion of an industrial robotic arm used in an assembly line, taking into account the forces and torques acting on it.
- **Variables**:
  - q: Vector of joint positions.
  - $\dot{q}$: Vector of joint velocities.
  - $\ddot{q}$: Vector of joint accelerations.
  - τ: Vector of torques applied at the joints.
  - $M(q)$: Inertia matrix.
  - $C(q, \dot{q})$: Coriolis and centrifugal forces matrix.
  - $G(q)$: Gravity vector.

4.2.3. Method Specification and Workflow

- **Dynamics Equations**: Use the Euler-Lagrange or Newton-Euler formulations to derive the equations of motion for the robotic arm.
  - Euler-Lagrange Formulation:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) - \frac{\partial L}{\partial q} = \tau$$

    where L is the Lagrangian representing the difference between kinetic and potential energies.
  - Newton-Euler Formulation:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = \tau$$

- **Simulation**: Integrate the equations of motion to simulate the arm's behavior under various control strategies.
- **Control**: Design control laws (like PID controllers, feedforward control, computed torque control) to achieve desired motion trajectories.

4.2.4. Strengths and Limitations

- **Strengths**:
  - Provides a detailed and accurate model of the robot's physical behavior.
  - Enables the design of efficient and responsive control systems.
- **Limitations**:

- Complex and highly nonlinear equations can be challenging to solve in real-time.
- Requires precise knowledge of all system parameters, which may not always be available or may change over time.

4.2.5. Common Problems in Application

- Uncertainties in model parameters like inertia and friction can lead to control inaccuracies.
- External disturbances and unmodeled dynamics can degrade performance.

4.2.6. Improvement Recommendations

- Use model identification techniques to refine the parameters in the dynamic model.
- Implement robust or adaptive control strategies to handle model uncertainties and external disturbances.

4.2.7. Discussion on Robotics Dynamics

The dynamics of robotics is a critical aspect that intersects with various disciplines, such as mechanical engineering, control theory, and computer science. It involves the application of classical mechanics to predict the behavior of robots, which is essential for designing control systems that can accurately guide their movements.

The dynamic model's complexity can vary significantly based on the robot's structure. For instance, a simple two-link planar arm has a much simpler model than a humanoid robot. Advanced robots require sophisticated models that account for the interplay between dynamics and control systems, especially when interacting with complex or uncertain environments.

In practical applications, the dynamics of robotics is often coupled with real-time sensing and feedback mechanisms to allow the robot to interact with its surroundings adaptively. This integration is pivotal in modern robotics applications, such as autonomous vehicles, drones, and interactive robotic systems.

General Robot System Dynamics

- **State** $x = (q, \dot{q}) \in \mathbb{R}^{2n}$
  – joint positions $q \in \mathbb{R}^n$
  – joint velocities $\dot{q} \in \mathbb{R}^n$
- **Controls** $u \in \mathbb{R}^n$ are the *torques* generated in each motor.
- The **system dynamics** are:

$$M(q)\,\ddot{q} + C(q,\dot{q})\,\dot{q} + G(q) = u$$

$M(q) \in \mathbb{R}^{n\times n}$	is positive definite intertia matrix (can be inverted $\rightarrow$ forward simulation of dynamics)
$C(q,\dot{q}) \in \mathbb{R}^n$	are the centripetal and coriolis forces
$G(q) \in \mathbb{R}^n$	are the gravitational forces
u	are the joint torques

4.3. Joint Space Control Method

4.3.1. Demo Case Study: Articulated Robotic Arm for Precision Assembly

- **Super Domain**: Robotics and Control Engineering
- **Type of Method**: Control Strategy

4.3.2. Problem Definition and Variables

- **Objective**: Control an articulated robotic arm to follow a desired trajectory with high precision for tasks requiring fine manipulation, such as in precision assembly lines.
- **Variables**:
  - q: Vector of joint angles or displacements.
  - $\dot{q}$: Vector of joint velocities.
  - $\ddot{q}$: Vector of joint accelerations.
  - τ: Vector of torques or forces applied to the robot's joints.
  - $q_{desired}$: Desired trajectory for each joint, defined as a function of time. We can get this with PID control or as spring damper system.

4.3.3. Method Specification and Workflow

- **Control Strategy**: Employ a feedback control loop to ensure that each joint of the robotic arm follows the desired trajectory as closely as possible.
- **Workflow**:
  i. Define the desired trajectory for each joint in terms of position, velocity, and acceleration.
  ii. Measure the current state of each joint.
  iii. Calculate the error between the desired and actual joint states.
  iv. Apply a control law (such as PID control) to compute the control input τ for each joint to correct the error.
  v. Adjust the control inputs based on dynamic models of the robotic arm to account for the interactions between joints due to the arm's mechanics.
  vi. Update the control inputs at each time step to continuously correct the trajectory.

4.3.4. Strengths and Limitations

- **Strengths**:
  - Direct control of each joint allows for straightforward implementation and interpretation.
  - Effective for systems where precise control over individual joint movements is necessary.
  - Well-suited for tasks with well-defined joint trajectories.
- **Limitations**:
  - May not consider the overall posture of the robot or the end-effector's position in Cartesian space.
  - Less efficient for tasks that require complex interaction with the environment or end-effector orientation control.

4.3.5. Common Problems in Application

- Coupling between joints can lead to complex dynamics that are challenging to control precisely.
- Variations in load or external disturbances can affect joint performance and lead to deviations from the desired trajectory.

4.3.6. Improvement Recommendations

- Implement feedforward control strategies alongside feedback control to anticipate and compensate for known system dynamics.
- Use real-time adaptive control to adjust for model uncertainties and external disturbances dynamically.

4.3.7. Joint Space Control Discussion

Joint space control is particularly advantageous when the precise motion of each joint is the primary objective, rather than the specific path the end-effector takes through Cartesian space. This approach is common in industrial robotics, where the robot's tasks are often broken down into a sequence of joint movements, and the end-effector's path is less critical than its final position or orientation.

In practice, joint space control requires accurate models of the robotic arm's dynamics, including the mass and inertia of the limbs, friction in the joints, and any flex in the structure. These models are used to predict how the robot will move in response to the applied torques and to design control laws that can achieve the desired movements.

Despite its widespread use, joint space control does have limitations, particularly when dealing with complex tasks that require the robot to interact with its environment in more sophisticated ways. For such tasks, Cartesian space control methods, which focus on the position and orientation of the end-effector in 3D space, may be more appropriate. However, for many applications, especially those with repetitive and well-defined movements, joint space control remains a robust and effective strategy.

Joint Space control

- Where could we get the desired $\ddot{q}^*$ from?
  Assume we have a nice smooth **reference trajectory** $q^{\text{ref}}_{0:T}$ (generated with some motion profile or alike), we can at each t read off the desired acceleration as

$$\ddot{q}^{\text{ref}}_t := \frac{1}{\tau}[(q_{t+1} - q_t)/\tau - (q_t - q_{t\text{-}1})/\tau] = (q_{t\text{-}1} + q_{t+1} - 2q_t)/\tau^2$$

What if we directly use desired reference acceleration?

Tiny errors in acceleration will **accumulate** greatly over time and this makes this an **unstable** approach!

4.4. Lagrangian Dynamics

4.4.1. Demo Case Study: Modeling a Bipedal Humanoid Robot

- **Super Domain**: Theoretical Mechanics and Robotics
- **Type of Method**: Analytical Dynamics Modeling

4.4.2. Problem Definition and Variables

- **Objective**: Develop a dynamic model for a bipedal humanoid robot to understand the forces and motions involved in tasks such as walking, running, or climbing.
- **Variables**:
  - q: Generalized coordinates representing the robot's configuration.
  - $\dot{q}$: Generalized velocities, the time derivatives of the coordinates.
  - L: Lagrangian, defined as the difference between the kinetic energy T and potential energy V of the system, $L = T - V$.
  - τ: Generalized forces acting on the system.

4.4.3. Method Specification and Workflow

- **Lagrangian Formulation**: Use the Lagrangian function to derive the equations of motion.
  - $\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = \tau_i$, for $i = 1$ to n, where n is the number of degrees of freedom.
- **Workflow**:
  i. Identify and label all degrees of freedom for the robot.
  ii. Define the kinetic and potential energy expressions in terms of the generalized coordinates and velocities.
  iii. Compute the partial derivatives of the Lagrangian with respect to each coordinate and velocity.
  iv. Apply the Lagrange equations to derive the second-order differential equations of motion.
  v. Solve the equations of motion for the generalized coordinates as functions of time to predict the robot's motion under various conditions.

4.4.4. Strengths and Limitations

- **Strengths**:
  - Provides a systematic approach to deriving equations of motion for complex, multi-body systems.
  - Avoids the need to directly deal with the internal forces and moments between components of the system.
- **Limitations**:
  - Can be algebraically complex, especially for systems with many degrees of freedom.
  - Requires accurate knowledge of all system parameters, such as masses, moments of inertia, and geometry.

4.4.5. Common Problems in Application

- Computational difficulty in solving the resulting differential equations, particularly for non-linear systems.
- Sensitivity to parameter variations, which can lead to significant errors in the predicted motions.

4.4.6. Improvement Recommendations

- Use computational tools like symbolic mathematics software for deriving and solving the Lagrange equations.
- Combine with experimental data for system identification to refine the model and improve accuracy.

4.4.7. Lagrangian Dynamics: Discussion

Lagrangian dynamics is an elegant and powerful framework that is particularly useful in robotics for analyzing systems where direct measurement of forces is challenging. By focusing on energy rather than forces, the Lagrangian approach simplifies the modeling process and is especially beneficial for understanding the motion of articulated robots like bipedal humanoids.

This approach allows for the decoupling of the robot's motion into individual components, which can be particularly advantageous when considering complex interactions within the robot's structure or between the robot and its environment. Additionally, it provides a foundation for developing advanced control strategies, such as those needed for balance and gait control in bipedal locomotion.

However, the complexity of the Lagrangian formulation increases significantly with the complexity of the robot's structure. For systems with a large number of interconnected parts, the equations can become unwieldy, and numerical methods may be required to solve them. Despite this, the insights gained from a Lagrangian-based analysis are invaluable for the design, optimization, and control of robotic systems.

4.5. Newton-Euler Dynamics

4.5.1. Demo Case Study: Real-Time Dynamics for Mobile Robotics

- **Super Domain**: Mechanics and Robotics
- **Type of Method**: Analytical Dynamics Modeling

4.5.2. Problem Definition and Variables

- **Objective**: Formulate the dynamics of a mobile robot for real-time applications, such as navigation and manipulation, by considering the forces and moments acting on the robot.
- **Variables**:
  - F: Vector of external forces applied to the robot.
  - τ: Vector of external torques applied to the robot.
  - m: Mass of the robot or robot component.
  - I: Inertia tensor of the robot or component.
  - a: Linear acceleration of the robot's center of mass.
  - α: Angular acceleration of the robot.
  - v: Linear velocity of the robot's center of mass.
  - ω: Angular velocity of the robot.

4.5.3. Method Specification and Workflow

- **Newton-Euler Equations**: Use Newton's second law for translation and Euler's equations for rotation to model the robot's dynamics.
  - Translation: $F = ma$
  - Rotation: $\tau = I\alpha + \omega \times (I\omega)$
- **Workflow**:
  i. Identify all forces and torques acting on the robot, including gravity, friction, and interaction with the environment.
  ii. Decompose the motion into translational and rotational components.
  iii. Apply Newton's second law to each body or component of the robot to find linear acceleration.
  iv. Apply Euler's equations to find angular acceleration.
  v. Integrate acceleration to find velocity and position for motion planning and control purposes.

4.5.4. Strengths and Limitations

- **Strengths**:
  - Directly applicable to rigid body dynamics and intuitive to understand.
  - Suitable for systems where forces and torques are readily measurable or estimable.
  - Facilitates the decoupling of translational and rotational dynamics, which simplifies analysis and control.
- **Limitations**:
  - Requires the assumption of rigid bodies, which may not be valid for robots with flexible parts.
  - Can become computationally expensive for systems with many interconnected bodies.

4.5.5. Common Problems in Application

- Difficulties arise in modeling systems with significant elasticity or deformability.
- Measurement errors in force and torque can lead to inaccuracies in the predicted dynamics.

4.5.6. Improvement Recommendations

- Implement recursive Newton-Euler algorithms for computational efficiency in systems with multiple interconnected bodies.
- Use sensor fusion techniques to improve the accuracy of force and torque measurements.

4.5.7. Newton-Euler Dynamics: Discussion

The Newton-Euler method is one of the fundamental approaches to modeling the dynamics of robotic systems. It's particularly useful in scenarios where the robot interacts with its environment dynamically, such as in manipulation tasks or when traversing uneven terrain.

For mobile robotics, where the dynamics can drastically affect navigation and stability, Newton-Euler dynamics provide a framework to ensure that all forces, including those resulting from collisions or interactions with various surfaces, are accounted for in the robot's control system.

In practice, while the Newton-Euler equations provide a rich description of motion, they often need to be supplemented with additional constraints or models to account for non-rigid behaviors, complex geometries, and real-world uncertainties. Advances in computational power and algorithms continue to expand the practical applicability of Newton-Euler dynamics in robotics.

4.6. Canonical Form for Articulated Rigid Bodies

4.6.1. Demo Case Study: Dynamics of a Multi-Joint Robotic Arm

- **Super Domain**: Robotics Dynamics
- **Type of Method**: Analytical Dynamics

4.6.2. Problem Definition and Variables

- **Objective**: To describe the motion of a multi-joint robotic arm, taking into account the interactions between its articulated rigid bodies.
- **Variables**:
  - q: Vector of joint displacements.
  - $\dot{q}$: Vector of joint velocities.
  - $\ddot{q}$: Vector of joint accelerations.
  - $M(q)$: Inertia matrix, depending on the configuration q.
  - $B(q)[\dot{q}]$: Coriolis matrix, a function of configuration q and velocities $\dot{q}$.
  - $C(q)[\dot{q}^2]$: Centrifugal matrix, a function of configuration q and the square of velocities $\dot{q}$.
  - $G(q)$: Gravity vector, dependent on the configuration q.
  - τ: Vector of external forces and torques applied to the joints.

4.6.3. Method Specification and Workflow

- **Canonical Form Equations**: The equations that describe the dynamics of articulated rigid bodies in their canonical form are:
  - $M(q)\ddot{q} + B(q)[\dot{q}] + C(q)[\dot{q}^2] + G(q) = \tau$
- **Workflow**:
  i. Determine the system's configuration and calculate the inertia matrix $M(q)$.
  ii. Compute the Coriolis and centrifugal forces using matrices $B(q)$ and $C(q)$, respectively.
  iii. Evaluate the gravity vector $G(q)$ based on the current configuration.
  iv. Use these terms to solve for the joint accelerations $\ddot{q}$ given the external forces/torques τ.
  v. Integrate the joint accelerations over time to obtain velocities and displacements for trajectory planning and control.

4.6.4. Strengths and Limitations

- **Strengths**:
  - Provides a complete and systematic approach to modeling the dynamics of complex articulated systems.
  - The canonical form separates different dynamic effects, which is useful for control and simulation purposes.
- **Limitations**:
  - The complexity of the matrices involved can lead to computationally intensive calculations, especially for systems with many degrees of freedom.
  - Requires accurate parameters for the inertia matrix and knowledge of external forces, which may not always be available.

4.6.5. Common Problems in Application

- Computational challenges in real-time applications due to the complexity of the matrices.
- Difficulties in measuring and estimating the parameters accurately in a real-world environment.

4.6.6. Improvement Recommendations

- Optimize the computational algorithms for real-time application, possibly using parallel computing techniques or simplifying assumptions.
- Combine with sensor data for parameter estimation and to account for unmodeled dynamics in control strategies.

4.6.7. Canonical Form for Articulated Rigid Bodies: Discussion

The canonical form for articulated rigid bodies is essential in robotics for understanding how the parts of a robotic system influence each other's motion. This formulation is particularly critical when designing controllers for robots that need to perform precise and complex tasks, as it directly relates the control inputs (forces/torques) to the resulting motion of the robot.

In practical scenarios, such as in the manufacturing industry, the canonical form helps in simulating the robot's behavior before actual deployment, which ensures safety and efficiency. It also provides a foundation for advanced control techniques, like model predictive control (MPC) or computed torque control, which can greatly enhance the performance of robotic systems.

The canonical form also serves as a bridge between the physical robot and digital simulation tools, enabling the development and testing of robotic systems within virtual environments. This greatly speeds up the design process and allows for the optimization of robot performance without the need for extensive physical prototypes.

Other representations in the literature

$$\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \tau$$

C is a vector with Coriolis plus centrifugal terms

$$\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \tau$$

C is a matrix with Coriolis plus centrifugal terms

4.7. Is Canonical Form for Articulated Rigid Bodies based on Newton-Euler dynamics?

Yes, the Canonical Form for Articulated Rigid Bodies is indeed based on Newton-Euler dynamics. The Newton-Euler approach provides the foundational principles for the forces and torques acting on each rigid body within a mechanical system. The Canonical Form for Articulated Rigid Bodies takes these principles and organizes them into a more structured and matrix-based representation that is particularly suited for complex, interconnected systems like robotic arms.

In essence, the Canonical Form encapsulates the Newtonian mechanics (Newton's laws for translational motion and Euler's laws for rotational motion) within a framework that is more convenient for computational modeling and control of articulated systems. It transforms the individual body dynamics described by Newton-Euler into a system-level description that accounts for the interactions between bodies due to their articulation.

To summarize:

- Newton-Euler dynamics provides the individual equations of force and torque balance for each rigid body.
- The Canonical Form aggregates these into a holistic model using matrices, making it suitable for systems with multiple interacting bodies, such as robots with several joints and links.

4.8. Recursive Newton-Euler Algorithm (RNEA) using Canonical Form

4.8.1. Demo Case Study: Dynamic Response of a Multi-Joint Robotic Manipulator

- **Super Domain**: Robotics Dynamics
- **Type of Method**: Computational Dynamics for Inverse Dynamics

4.8.2. Problem Definition and Variables

- **Objective**: To calculate the inverse dynamics of a multi-joint robotic manipulator efficiently, providing the joint torques needed for a given movement.
- **Variables**:
  - q: Vector of joint positions.
  - $\dot{q}$: Vector of joint velocities.
  - $\ddot{q}$: Vector of joint accelerations.
  - $M(q)$: Inertia matrix as a function of the joint configuration q.
  - $C(q)$: Coriolis and centrifugal forces matrix, dependent on the configuration q and joint velocities $\dot{q}$.
  - $G(q)$: Gravity vector, a function of the joint configuration q.
  - τ: Vector of joint torques resulting from external forces and moments.

4.8.3. Method Specification and Workflow

- **Algorithmic Approach**: The RNEA uses the kinematic chain of the robot to calculate joint torques through two recursive passes, one to compute velocities and accelerations and another for forces and torques.
- **Workflow**:
  i. **Initialization**: Set base velocities and accelerations to zero: $\dot{q}_0 = 0$, $\ddot{q}_0 = 0$.
  ii. **Forward Recursion**: Compute velocities and accelerations for each joint/link.
    - $\dot{v}_i = \dot{v}_{i-1} + \ddot{q}_i \times r_{i-1} + \dot{q}_i \times (\dot{q}_i \times r_{i-1})$
    - $\dot{\omega}_i = \dot{\omega}_{i-1} + \ddot{q}_i$
  iii. **Backward Recursion**: Compute joint forces and torques from the end-effector to the base.
    - $f_i = M_i \dot{v}_i + C_i(\dot{q}_i) \times \dot{q}_i + G_i$
    - $\tau_i = f_{i-1} \times r_{i-1} + f_i \times r_i + \tau_{ext,i}$
  iv. **Output**: The resulting τ vector represents the joint torques needed to produce the desired motion.

4.8.4. Strengths and Limitations

- **Strengths**:
  - Algorithmic efficiency makes it suitable for real-time applications.
  - Can handle complex, multi-joint robotic systems.
- **Limitations**:
  - Relies on accurate modeling of the robot's physical parameters.
  - Assumes rigid bodies, which may not account for all dynamics in flexible or soft robotic systems.

4.8.5. Common Problems in Application

- Errors in parameter estimation can lead to inaccurate torque computations.
- Real-world factors such as joint friction or unmodeled dynamics can affect the results.

4.8.6. Improvement Recommendations

- Enhance parameter estimation through sensor data integration.
- Implement control strategies that can compensate for real-world discrepancies from the modeled dynamics.

A recursive algorithm for articulated robots

- The Recursive Newton-Euler Algorithm (RNEA) gives us an iterative way to compute Inverse Dynamics:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \tau$$

$$RNEA(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \tau$$

- Without requiring to work out the mass matrix. However we still need M for forward dynamics

$$\ddot{\mathbf{q}} = \mathbf{M}(\mathbf{q})^{-1}(-\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{G}(\mathbf{q}) + \tau)$$

28

Can RNEA be used to compute the mass matrix?

- A trick is to compute M column by column by setting:

$$\mathbf{g} = 0 \quad \dot{\mathbf{q}} = 0 \quad \ddot{q}_i = 1 \quad \ddot{q}_j = 0, \forall j \neq i$$

- In such a case RNEA return the ith column of M
- As a result, computing inverse dynamics is faster than forward dynamics:

- $O(n^2)$ for computing *direct dynamics*,
- $O(n)$ for computing *inverse dynamics*.

4.9. Why the torque in backward recursion of RNEA add i-1 term?

In the backward recursion step of the Recursive Newton-Euler Algorithm (RNEA), the torque for each joint is calculated by considering both the force applied to the current joint/link and the forces from the subsequent joints/links. The reason for including the f_{i-1} term (forces from the previous joint) when calculating τ_i is to account for the dynamic effects that propagate along the kinematic chain of the robot.

When calculating the torque at joint i, you must consider not only the direct forces and moments acting on that joint but also the effects of all the forces and moments acting on all the links that follow joint i in the kinematic chain. This is because the motion of each link affects the motion of all subsequent links. Therefore, the torques at each joint are the result of both the local forces/moments and the transmitted forces/moments from all subsequent joints and links.

The term $f_{i-1} \times r_{i-1}$ represents the moment due to the force at the previous joint/link acting at a distance r_{i-1} from the current joint. This term ensures that the torques calculated include these transmitted forces, which are crucial for accurately determining the net torque required at each joint to achieve the desired motion.

This recursive accumulation of forces and moments from the end-effector back to the base is what allows RNEA to be so efficient and accurate for calculating the inverse dynamics of articulated robotic systems.

5. Controller

5.1. First-Order Error Dynamics (PD control)

5.1.1. Demo Case Study: Velocity Control of a Mass-Damper System

- **Super Domain**: Control Systems, Dynamics
- **Type of Method**: Error Dynamics Analysis

5.1.2. Problem Definition with Variables Notations

- **Objective**: Analyze the error dynamics of a system to predict how the system responds to a change from a desired state, focusing on velocity control.
- **Variables**:
  - θ_e: Error in position or angle.
  - $\dot{\theta}_e$: Error in velocity.
  - m: Mass of the object.
  - b: Damping coefficient.
  - k: Spring constant.
  - t: Time constant, where $t = \frac{b}{k}$.

5.1.3. Assumptions

- The system behaves according to linear dynamics.
- No external forces are acting on the system except for the spring and damper.

5.1.4. Method Specification and Workflow

- **Standard Form of First-Order ODE**:
  - The error dynamics are described by a first-order ordinary differential equation (ODE):

$$\dot{\theta}_e(t) + \frac{k}{b}\theta_e(t) = 0$$

- **Time Constant**:
  - The time constant t is a measure of how quickly the system responds to changes in the error. It's defined as the time it takes for the error to decrease by a factor of e (Euler's number) from its initial value.

5.1.5. Comment: Strengths and Limitations

- **Strengths**:
  - First-order error dynamics are relatively simple to analyze and provide insights into system behavior.
  - The time constant gives a straightforward measure of the system's responsiveness.
- **Limitations**:
  - Assumes a linear relationship, which may not hold for all real-world systems, especially those with nonlinear characteristics or significant external disturbances.

5.1.6. Common Problems When Applying the Method

- Oversimplification of complex dynamics can lead to inaccurate predictions.
- Damping ratio and natural frequency are not considered, which are important in second-order systems.

5.1.7. Improvement Recommendations

- For systems where higher-order dynamics are significant, use higher-order differential equations for analysis.
- Incorporate adaptive control techniques to handle systems with varying parameters or non-linear behavior.

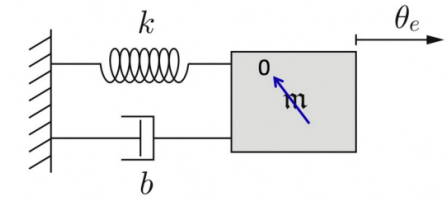
5.1.8. Discussion on First-Order Error Dynamics

The first-order error dynamics provide a basic understanding of how a system will respond to errors in velocity. The damping coefficient b plays a crucial role in determining the rate at which the error will decay. A larger b results in a faster error decay, which can be desirable for quickly stabilizing a system but may lead to higher energy consumption or more aggressive control actions.

In practical applications, the first-order error dynamics can be used to design feedback controllers that effectively reduce velocity errors, such as in automated manufacturing systems where precise speed control is necessary. It's also useful in robotics for controlling the velocity of actuators or motors.

The concept of time constant t is particularly useful for tuning controllers in a first-order system, providing a direct relationship between the system parameters and the desired speed of response. However, for more complex systems, this first-order model may need to be expanded or used in conjunction with other control strategies to achieve the desired control performance.

First-order Error Dynamics



0

$$m\ddot{\theta}_e + b\dot{\theta}_e + k\theta_e = 0$$

$$\dot{\theta}_e(t) + \frac{k}{b}\theta_e(t) = 0$$

standard first-order form

time constant

$$t = b/k$$

$$\dot{\theta}_e(t) + \frac{1}{t}\theta_e(t) = 0$$

5.2. Second-Order Error Dynamics (PID control)

Case Study: Modeling the response of a system with acceleration considered, typically seen in mechanical systems with mass, damping, and spring constants.

- **Super Domain**: Control Systems, specifically in the context of mechanical systems with mass-spring-damper assemblies.
- **Method Level/Type**: Second-order linear ordinary differential equation representing physical systems.

5.2.1. Problem Definition:

- Variables:
  - $\theta_e(t)$: Error in position at time t.
  - $\dot{\theta}_e(t)$: Error in velocity at time t.
  - $\ddot{\theta}_e(t)$: Error in acceleration at time t.
  - k: Spring constant.
  - b: Damping coefficient.
  - m: Mass of the object.
  - ω_n: Natural frequency of the system.
  - ζ: Damping ratio of the system.

5.2.2. Assumptions:

- The system is modeled as a linear second-order system.
- No external forces acting on the system (homogeneous equation).
- The system parameters m, b, and k are constant over time.

5.2.3. Method Specification and Workflow:

- Standard form of the second-order differential equation:

$$m\ddot{\theta}_e + b\dot{\theta}_e + k\theta_e = 0$$

- This can be rewritten using the standard form variables as:

$$\ddot{\theta}_e + 2\zeta\omega_n\dot{\theta}_e + \omega_n^2\theta_e = 0$$

- The solution involves calculating the system's natural frequency ω_n and damping ratio ζ, which are given by:

$$\omega_n = \sqrt{\frac{k}{m}}, \quad \zeta = \frac{b}{2\sqrt{km}}$$

5.2.4. Strengths and Limitations:

- **Strengths**:
  - Provides a clear understanding of how the system responds over time.
  - Useful in designing controllers to achieve desired transient response characteristics.
- **Limitations**:
  - Assumes a linear system, which may not hold for all physical systems.
  - Does not account for external forces or non-linearities in the system.

5.2.5. Common Problems When Applying the Method:

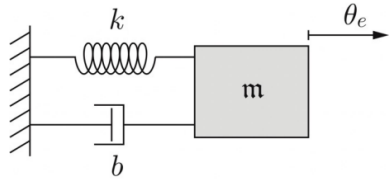
- Estimating accurate values for the physical parameters m, b, and k can be challenging.
- Linear models might not accurately represent the behavior of real-world systems that exhibit non-linear dynamics.

5.2.6. Improvement Recommendation:

- For systems that do not strictly adhere to the assumptions of linearity, incorporate non-linear dynamics into the model.
- Use experimental data to refine estimates of m, b, and k for better model fidelity.
- Explore the inclusion of PID controllers to manage complex dynamics that involve both position and velocity error responses.

5.2.7.

Second-order Error Dynamics



$$\ddot{\theta}_e(t) + \frac{b}{\mathrm{m}}\dot{\theta}_e(t) + \frac{k}{\mathrm{m}}\theta_e(t) = 0$$

natural frequency damping ratio

$$\omega_n = \sqrt{k/\mathrm{m}} \qquad \zeta = b/(2\sqrt{k\mathrm{m}})$$

$$\ddot{\theta}_e(t) + 2\zeta\omega_n\dot{\theta}_e(t) + \omega_n^2\theta_e(t) = 0$$

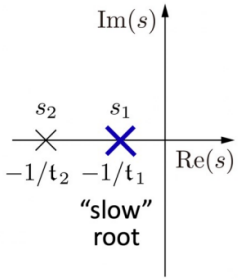
standard second-order form

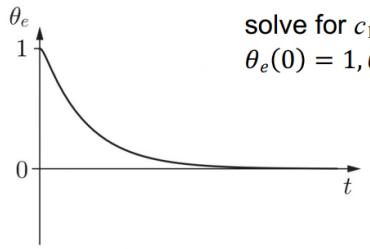
Overdamped behaviour

$\zeta > 1$: Overdamped

$$\theta_e(t) = c_1 e^{s_1 t} + c_2 e^{s_2 t}$$

$$s_1 = -\zeta\omega_n + \omega_n\sqrt{\zeta^2 - 1}$$
$$s_2 = -\zeta\omega_n - \omega_n\sqrt{\zeta^2 - 1}$$



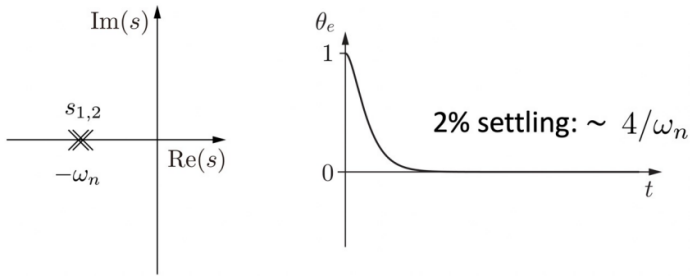


solve for c_1 and c_2 using
$\theta_e(0) = 1, \dot{\theta}_e(0) = 0$

Critically damped behaviour

$\zeta = 1$: Critically damped

$\theta_e(t) = (c_1 + c_2 t)e^{-\omega_n t}$ $\quad s_{1,2} = -\omega_n$



2% settling: ~ $4/\omega_n$

Underdamped behaviour

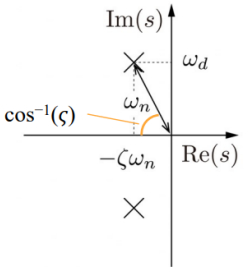
$\zeta < 1$: Underdamped

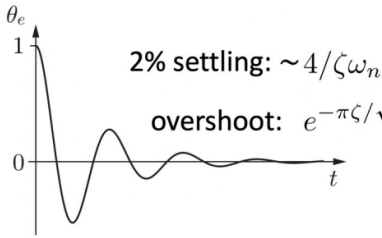
$\theta_e(t) = (c_1 \cos \omega_d t + c_2 \sin \omega_d t)\, e^{-\zeta\omega_n t}$

damped natural frequency

$\omega_d = \omega_n \sqrt{1-\zeta^2}$

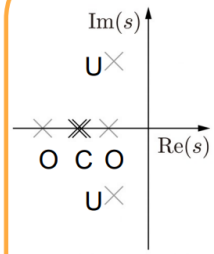
$s_{1,2} = -\zeta\omega_n \pm j\omega_d$

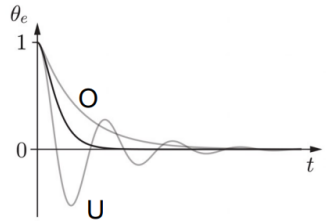




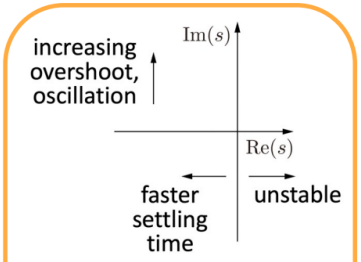
2% settling: ~$4/\zeta\omega_n$

overshoot: $e^{-\pi\zeta/\sqrt{1-\zeta^2}} \times 100\%$





second-order systems



time response relative to root locations for general systems

5.3. Design of PID Controllers

5.3.1. Demo Case Study: Controlling a Robotic Arm's Position

- **Super Domain**: Digital System and Digital Controllers
- **Type of Method**: Control Algorithm

5.3.2. Problem Definition and Variables

- **Objective**: To maintain the robotic arm's position at a desired setpoint.
- **Variables**:
  - $e(t)$: Error signal, difference between desired setpoint and current position.
  - K_p: Proportional gain.
  - K_i: Integral gain.
  - K_d: Derivative gain.
  - $u(t)$: Control signal to the robotic arm.

5.3.3. Assumptions

- The system is linear and time-invariant.
- Disturbances and noise are minimal or can be neglected.

5.3.4. Method Specification and Workflow

1. **Proportional Control**: $u(t) = K_p e(t)$
   - Directly proportional to the error.
2. **Integral Control**: Adds integral term $K_i \int e(t)dt$
   - Addresses accumulated error over time.
3. **Derivative Control**: Adds derivative term $K_d \frac{de(t)}{dt}$
   - Predicts future error based on its rate of change.
4. **Combined PID Control**: $u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}$

5.3.5. Strengths and Limitations

- **Strengths**:
  - Simple to understand and implement.
  - Effective in a variety of systems and conditions.
  - Adjustable gains for system tuning.
- **Limitations**:
  - Performance can degrade in the presence of nonlinearities and disturbances.
  - Requires careful tuning of parameters.
  - May lead to instability if not properly configured.

5.3.6. Common Problems in Application

- Overshoot and undershoot due to improper gain settings.

- Oscillations if the derivative term is not appropriately tuned.
- Steady-state error if the integral term is insufficient.

5.3.7. Improvement Recommendations

- Implement adaptive PID control where gains adjust based on system performance.
- Combine with other control strategies for handling non-linearities (e.g., feedforward control).
- Use modern tuning methods like Ziegler-Nichols for optimal parameter setting.

5.3.8. PID Gain Tuning

There are a number of methods for tuning a PID controller to get a desired response. Below is a summary of how increasing each of the control gains affects the response:

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Minor change	Decrease	Decrease	No effect	Improve if K_d small

- **Rise Time**: Rise time refers to the time it takes for the system's response to go from a certain percentage of the steady-state value (commonly 10%) to another percentage (commonly 90%) for the first time. It's an indicator of how quickly the system responds to a change in input.
- **Overshoot**: Overshoot is the extent to which the system's response exceeds its steady-state value. It's typically measured as a percentage of the final steady-state value. High overshoot can be indicative of a system that's too responsive and thus potentially unstable.
- **Settling Time**: Settling time is the time taken for the system's response to remain within a certain percentage (commonly 2% or 5%) of the steady-state value after a disturbance or a change in input. It's a measure of how quickly the system settles into its final stable state.
- **Steady-State Error**: Steady-state error is the difference between the system's steady-state output and the desired output. It measures the accuracy of the system in achieving the desired output after the transient effects have died out. A well-tuned system should have a steady-state error that is as small as possible.
- **Stability**: In control systems, stability refers to the ability of a system to converge to a steady state after a disturbance or a change in input. A stable system's output will not diverge over time. Instability, conversely, may be indicated by oscillations that increase in amplitude over time. Stability is a fundamental requirement for any control system to be reliable and predictable in its operation.

- ❏ k_d term predicts system behaviour in *one tick*, which gives a control effort with the anticipation of the change during the next sampling time.
- ❏ In theory, given any k_p gain, there is always a k_d gain that can ensure critical damping of the response. However, due to the noise and delay of velocity, k_d cannot to be too large otherwise noise in amplified. Therefore, k_p gain can't be too large either.

5.3.9. Design of PID Controllers: Discussion and Behavioral Table

The design of PID controllers is a critical step in ensuring that control systems are both responsive and stable. The balance of P, I, and D components dictates the behavior of the system's natural frequency and damping ratio. Below is a table summarizing the general effects of increasing each PID component on the natural frequency (ω_n) and damping ratio (ζ):

Controller Parameter	Increase in K_p	Increase in K_i	Increase in K_d
Natural Frequency	Increases	Increases	Minor effect or decreases
Damping Ratio	Decreases	Increases	Increases

*Note: The exact impact on natural frequency and damping ratio can vary depending on the specific system dynamics.

In PID tuning, the aim is to achieve a desired transient response (speed of response, overshoot) and steady-state accuracy. Proportional gain K_p improves the response speed but may reduce system stability, leading to oscillations. Integral gain K_i eliminates steady-state error but may lead to slower response and increased overshoot. Derivative gain K_d anticipates future errors, improving stability and reducing overshoot but can be sensitive to measurement noise.

The design process often involves trade-offs, and the use of tuning methods or heuristic rules can help find an optimal balance that satisfies performance criteria. Additionally, simulation tools can provide a safe environment to test and refine PID settings before applying them to the actual system.

5.4. Inverse Dynamics Control

5.4.1. Demo Case Study: Robotic Arm Trajectory Tracking

- **Super Domain**: Robotic Control Systems
- **Type of Method**: Inverse Dynamics Control Algorithm

5.4.2. Problem Definition and Variables

- **Objective**: To compute joint torques τ that achieve a desired acceleration $\ddot{q}^d$ for trajectory tracking.
- **Variables**:
  - q: Vector of actual joint positions.
  - $\dot{q}$: Vector of actual joint velocities.

- $\ddot{q}$: Vector of actual joint accelerations.
- $q^r(t)$: Reference trajectory for joint positions at time t.
- $\dot{q}^r(t)$: Reference trajectory for joint velocities.
- $\ddot{q}^r(t)$: Reference trajectory for joint accelerations.
- K_p: Proportional gain matrix.
- K_v: Derivative gain matrix.
- h: Vector of non-linear dynamics terms including Coriolis, centrifugal, and gravitational forces.

5.4.3. Method Specification and Workflow

- **Control Law**: The control torque τ is calculated to achieve the desired joint acceleration $\ddot{q}^d$ by compensating for the robot dynamics and tracking the reference trajectory.
- **Workflow**:
  i. Calculate the error in position $e = q - q^r$ and velocity $\dot{e} = \dot{q} - \dot{q}^r$.
  ii. Compute the desired acceleration $\ddot{q}^d$ using feedback control:

$$\ddot{q}^d = \ddot{q}^r - K_p e - K_v \dot{e}$$

  iii. Apply the inverse dynamics formula to compute the control torques:

$$\tau = M(q)\ddot{q}^d + h$$

  where h includes terms for Coriolis, centrifugal, and gravitational forces.
  iv. Send the computed torques τ to the robot's actuators.

5.4.4. Strengths and Limitations

- **Strengths**:
  - Precision in trajectory tracking due to the model-based control strategy.
  - Effective compensation for the robot's own dynamics.
- **Limitations**:
  - Highly reliant on an accurate dynamic model of the robot.
  - Requires precise measurement of joint positions and velocities.

5.4.5. Common Problems in Application

- Model inaccuracies leading to tracking errors.
- Uncertainties in measuring joint velocities and accelerations.

5.4.6. Improvement Recommendations

- Implement model identification techniques to refine the dynamic model.
- Use sensor fusion to improve the accuracy of state measurements.

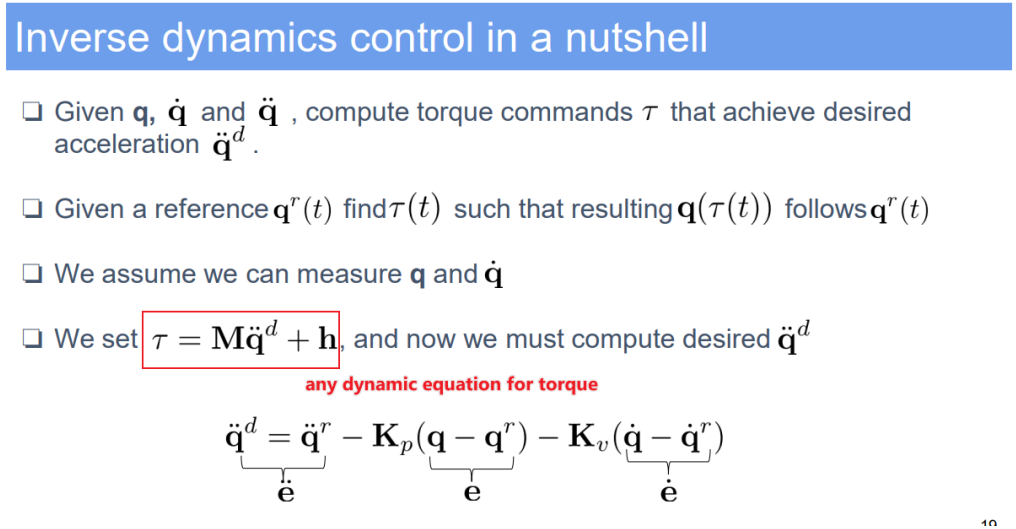
5.4.7. Inverse Dynamics Control: Discussion

Inverse dynamics control is a robust approach for commanding robotic systems to follow a desired trajectory. It involves calculating the torques that must be applied at the robot's joints to produce the specified motion, considering the robot's actual dynamics.

This method provides a way to design control laws that anticipate the effects of the robot's mass, friction, and other physical properties, allowing the system to move smoothly and accurately. However, its effectiveness is highly dependent on the accuracy of the robot model used to calculate the dynamics. Any discrepancy between the model and the actual robot can result in performance degradation.

The inclusion of feedback gains K_p and K_v helps to correct for tracking errors and to dampen oscillations, respectively. Adjusting these gains allows for fine-tuning the control system's responsiveness and stability.

For practitioners, this approach necessitates a deep understanding of the robot's physical makeup and dynamics. Continuous monitoring and adjustment might be needed to maintain optimal performance, especially in changing environmental conditions or as the robot experiences wear and tear over time.



- Option 1: compute corresponding $\mathbf{q}^r(t)$ then apply ID control
  - Issue 1: this is the inverse geometry problem, non-linear problem with infinity of solutions
  - Issue 2: Tracking $\mathbf{q}^r(t)$ is **sufficient** but not necessary to track $\mathbf{x}^r(t)$

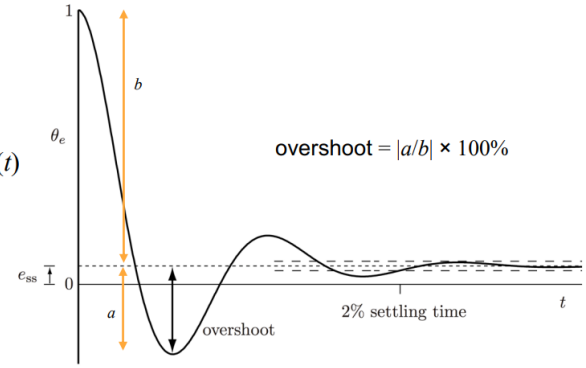
    This means that perturbations that affect $\mathbf{q}^r(t)$ but not the Forward Geometry FG(**q**) are rejected

5.5. Concept and definition in control

What is "Dynamic Response"?

For motion control,

reference: $\theta_d(t)$
actual: $\theta(t)$
error: $\theta_e(t) = \theta_d(t) - \theta(t)$



Unit step error response:
$\theta_e(t)$ starting from $\theta_e(0) = 1$

Steady-state error response: e_{ss}

Transient error response:
overshoot, settling time

Concept: Error Response

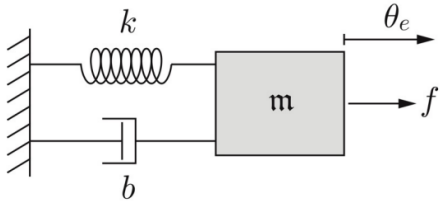
System dynamics, feedback controllers, and error response are often modeled by **linear ordinary differential equations**.

The simplest linear ODE exhibiting overshoot is second order, e.g.,

$$\mathfrak{m}\ddot{\theta}_e + b\dot{\theta}_e + k\theta_e = f$$

or, if $f = 0$,

$$\ddot{\theta}_e + \frac{b}{\mathfrak{m}}\dot{\theta}_e + \frac{k}{\mathfrak{m}}\theta_e = 0$$



k and b depend on the control law

A more general p[th]-order linear ODE:

$$a_p\theta_e^{(p)} + a_{p-1}\theta_e^{(p-1)} + \cdots + a_2\ddot{\theta}_e + a_1\dot{\theta}_e + a_0\theta_e = c \quad \text{nonhomogenous}$$

$$a_p\theta_e^{(p)} + a_{p-1}\theta_e^{(p-1)} + \cdots + a_2\ddot{\theta}_e + a_1\dot{\theta}_e + a_0\theta_e = 0 \quad \text{homogeneous}$$

$$\theta_e^{(p)} + a'_{p-1}\theta_e^{(p-1)} + \cdots + a'_2\ddot{\theta}_e + a'_1\dot{\theta}_e + a'_0\theta_e = 0$$

$$\theta_e^{(p)} = -a'_{p-1}\theta_e^{(p-1)} - \cdots - a'_2\ddot{\theta}_e - a'_1\dot{\theta}_e - a'_0\theta_e$$

Defining a state vector $x = (x_1, x_2, ..., x_p)$, you can write the p^{th}-order ODE as p first-order ODEs (a vector ODE).

$$x_1 = \theta_e,$$
$$x_2 = \dot{x}_1 = \dot{\theta}_e,$$
$$\vdots \quad \vdots$$
$$x_p = \dot{x}_{p-1} = \theta_e^{(p-1)}$$

$$\dot{x}(t) = Ax(t) \rightarrow x(t) = e^{At}x(0)$$

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ -a'_0 & -a'_1 & -a'_2 & \cdots & -a'_{p-2} & -a'_{p-1} \end{bmatrix} \in \mathbb{R}^{p\times p}$$

$$\dot{x}(t) = Ax(t) \rightarrow x(t) = e^{At}x(0)$$

If $\mathrm{Re}(s) < 0$ for all eigenvalues s of A, then the error dynamics are **stable** (the error decays to zero).

The eigenvalues are the roots of the **characteristic equation**

$$\det(sI - A) = s^p + a'_{p-1}s^{p-1} + \cdots + a'_2s^2 + a'_1s + a'_0 = 0$$

Necessary conditions for stability: each $a'_i > 0$.

These necessary conditions are also **sufficient** for first- and second-order systems.

5.5.1. What is $Re(s)$

In the context of control theory and differential equations, $\mathrm{Re}(s)$ refers to the real part of a complex number s. When discussing system stability, particularly for linear time-invariant (LTI) systems, the stability is often determined by the location of the poles of the system's transfer function in the complex plane.

A pole is a value of s (which can be complex) that makes the transfer function become unbounded. The transfer function is derived from the characteristic equation of the system's differential equation. The characteristic equation is obtained by applying the Laplace transform to the differential equation and setting the Laplace transform of the output to zero.

For a system to be stable, all poles must have negative real parts, meaning that $\mathrm{Re}(s) < 0$ for all eigenvalues s of A, where A is the system matrix. If $\mathrm{Re}(s)$ is positive for any pole, the system will exhibit exponential growth in its response, leading to instability. If $\mathrm{Re}(s)$ is zero for any pole, the system may be marginally stable or unstable depending on the multiplicity of the poles and the specific system characteristics.

Therefore, $\mathrm{Re}(s) < 0$ is a condition for the asymptotic stability of the system, ensuring that any perturbations or errors in the system's response will decay over time, and the system will return to its equilibrium state.

5.6. Digital PID control

Digital Implementation

Controls are often implemented in computer-based systems or by digital computation, e.g. micro-controllers, DSP, FPGA etc.
A digital control system only 'sees' the sensory information and command the control action at times, at a constant time interval.

The continuous PID law $\boldsymbol{u}(t) = k_p e + k_d \dot{e} + k_I \int e \, dt$

Can be rewritten with appropriately adjusted coefficients as:

$$\boldsymbol{u}(k) = k_p e(k) + k_d \dot{e}(k) + k_i \sum e(i), i = 0, \ldots, k$$

Using backward Euler method:

$\dot{e}(k) = \frac{[e(k) - e(k-1)]}{T}$ (usually, derivative terms are filtered)

$\int e \, dt = T \sum e(i)$, note, in k[th] control loop, range of i is: $i = 0, \ldots, k$

PID in continuous time

$$\boldsymbol{u}(t) = k_p e + k_d \dot{e} + k_I \int e \, dt$$

PID in discrete time

$$\boldsymbol{u}(k) = k_p e(k) + k_d \frac{[e(k) - e(k-1)]}{T} + k_I T \sum e(i)$$

$$\boldsymbol{u}(k) = k_p e(k) + k_d \dot{e}(k) + k_i \sum e(i), i = 0, \ldots, k$$

5.7. Feedback and feedforward control

Feedback and feedforward control are two fundamental approaches to system control in automation and robotics:

- **Feedforward Control (Open-loop control)**:
  - This type of control does not use feedback to determine if its output has achieved the desired goal of the input command or process set point.
  - It operates on the basis of pre-set conditions. For example, in a feedforward control system, a specific input will result in a known output without the system checking the results.
  - In the context of the joint control you've provided, the joint velocity is set to a desired value ($\dot{\theta}_d(t)$) directly. This method assumes that the system's behavior is predictable enough that feedback isn't necessary.
  - However, without feedback, there's no compensation for disturbances or variations in the system's behavior, so the actual output might differ from the expected output.
- **Feedback Control (Closed-loop control)**:
  - In contrast to feedforward control, feedback control involves real-time acquisition of data related to the output or the process condition.
  - The control action is based on the current state of the output and the desired output. This means that any error in the system (difference between the actual and desired output) is used to make adjustments to reach the desired goal.
  - For example, $\dot{\theta}(t)$ is adjusted based on the function $f(\theta_d(t), \theta(t))$, which accounts for the actual position ($\theta(t)$) and the desired position ($\theta_d(t)$) of the joint.
- **Proportional-Integral (PI) Feedback Control**:
  - This combines both proportional control (P) and integral control (I) to adjust the controller output.
  - The proportional term (K_p) produces an output value that is proportional to the current error value. It provides a control action to counteract the present value of the error.
  - The integral term (K_i) is concerned with the accumulation of past error values and introduces a control action based on the sum of the errors over time, helping to eliminate steady-state errors.

- In your control equation, $\dot{\theta}(t)$ is adjusted by adding a term that accounts for the current error ($K_p\theta_e(t)$) and the integral of the error over time ($K_i \int \theta_e(t)dt$), providing a balance of immediate correction with historical error correction.

The choice between feedforward and feedback control, or a combination thereof, depends on the system requirements, the predictability of the system dynamics, and the presence of disturbances. Feedforward control is typically faster but less accurate, while feedback control can be more accurate and robust but might introduce a delay in the response.

Concept: Feedback vs. Feedforward

For a single joint with the joint velocity as the control:

- **Open-loop (feedforward) control**: $\dot{\theta}(t) = \dot{\theta}_d(t)$
- **Closed-loop (feedback) control**: $\dot{\theta}(t) = f(\theta_d(t), \theta(t))$
- **FF + Proportional-Integral (PI) FB control**:

$$\dot{\theta}(t) = \dot{\theta}_d(t) + K_p\theta_e(t) + K_i \int_0^t \theta_e(t)\, dt \ , \ K_p, K_i \geq 0$$

  - *reduces to FF control if* $K_p, K_i = 0$
  - *if no FF term:* **P control** *when* $K_i = 0$, **I control** *when* $K_p = 0$

5.8. Apply contact force

A very short note on contacts (for the lab)

❑ We have seen that

$$\tau = \mathbf{M}\ddot{\mathbf{q}} + \mathbf{h}$$

❑ What if we introduce contacts?

We can write

$$\tau = \mathbf{M}\ddot{\mathbf{q}} + \mathbf{h} + \mathbf{J}^T\mathbf{f}_c$$

Where $\mathbf{f}_c$ is a 6D contact force. To control your robot for lifting the cube, you can set a desired $\mathbf{f}_c$ on both effectors and use the control laws we have used before to compute the appropriate torques.

If equiped with a force sensor, you could also implement a PI control to track the error accurately.

5.9. Optimal Control

Optimal Control is a mathematical framework aimed at finding a control policy that minimizes a certain cost function over time for a given dynamic system. The cost function typically includes terms representing the state and control effort, and may also incorporate a terminal cost evaluating the final state.

Objective:

- Minimize the path cost integral plus terminal cost, formally represented as:

$$\min_{X,U} \int_0^T l(x(t), u(t))dt + l_T(x(T))$$

Constraints:

- The system must adhere to its dynamic model, represented by the state derivative $\dot{x}(t)$, which is a function of the current state $x(t)$ and control input $u(t)$:

$$\dot{x}(t) = f(x(t), u(t))$$

Variables:

- X and U are the state and control vectors, respectively, which are functions of time t.
- $x(t)$ represents the state vector at time t, mapping from real numbers to an n-dimensional state space.
- $u(t)$ denotes the control input vector at time t, mapping to an m-dimensional control space.
- The terminal time T is fixed, marking the endpoint for the optimization horizon.

Discussion:
Optimal control problems are central to many engineering disciplines, particularly in robotics and automation. They provide a rigorous method for designing control systems that can perform complex tasks efficiently. However, solving these problems can be computationally challenging, especially for systems with high dimensionality or complex dynamics.

Understanding both the theory and practical application through labs is essential, as optimal control often requires a balance between theoretical knowledge and practical tuning of control parameters.

The terminal cost $l_T(x(T))$ is particularly significant as it allows incorporating goals or final conditions into the optimization process, such as reaching a target state or minimizing energy use by the terminal time T.

To implement an optimal control policy, one must understand the dynamic model of the system, which describes how the system evolves over time under various control inputs. This is crucial in applications like trajectory planning for autonomous vehicles, energy-efficient operation of systems, and robotic manipulator control, where the goal is to achieve desired outcomes while minimizing some notion of cost.

6. Other

6.1. Gradient Descent Approach in Robotics

6.1.1. Demo Case Study: Optimizing Path Planning for an Autonomous Vehicle

- **Super Domain**: Optimisation in Robotics
- **Type of Method**: Numerical Optimization Technique

6.1.2. Problem Definition and Variables

- **Objective**: To find the optimal path for an autonomous vehicle to navigate from a starting point to a destination while minimizing a cost function.
- **Variables**:
  - $\mathbf{x}$: Vector representing the state of the vehicle (position, velocity, etc.).
  - $f(\mathbf{x})$: Cost function to be minimized, e.g., distance, time, or energy consumption.

6.1.3. Assumptions

- The cost function is differentiable with respect to the state variables.
- The environment is static and can be represented as a known cost landscape.

6.1.4. Method Specification and Workflow

1. **Initialization**: Start with an initial guess for the state vector $\mathbf{x}$.
2. **Gradient Calculation**: Compute the gradient of the cost function, $\nabla f(\mathbf{x})$, representing the direction of steepest ascent.
3. **Update Step**: Update the state vector in the direction of the negative gradient to move towards the minimum.
   - Update rule: $\mathbf{x}_{new} = \mathbf{x}_{old} - \alpha \nabla f(\mathbf{x}_{old})$, where α is the learning rate.
4. **Iteration**: Repeat the gradient calculation and update steps until convergence criteria are met (e.g., changes in cost function are below a threshold).

6.1.5. Strengths and Limitations

- **Strengths**:
  - Widely applicable to a variety of optimization problems in robotics.
  - Conceptually simple and easy to implement.
  - Can be adapted to different environments and cost functions.
- **Limitations**:
  - Convergence to the global minimum is not guaranteed, especially in non-convex landscapes.
  - Choice of learning rate α is critical and can affect the speed and success of convergence.
  - Computationally intensive for high-dimensional problems.

6.1.6. Common Problems in Application

- Getting stuck in local minima, missing the global minimum.
- Oscillations or divergence if the learning rate is too high.
- Slow convergence if the learning rate is too low or if the gradient is shallow.

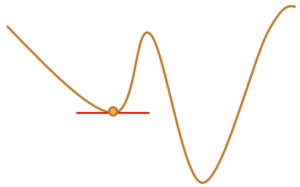
6.1.7. Improvement Recommendations

- Use adaptive learning rate techniques to adjust α dynamically.
- Implement advanced variants like stochastic gradient descent for large-scale problems.
- Combine with other optimization strategies (e.g., genetic algorithms, simulated annealing) to escape local minima.

Gradient descent => find the minimum of a function

❑ Here distance between current position / target can be used as such function

❑ Trying to find the **global minima** will not always work…



6.2. Assumption: sensor accurate reading

❑ We also assume motors are equipped with accurate **position** sensors (ie we know **q** accurately)

If we remove this assumption we need Kalman Filter.