Robotics Control Notes

Devodita Chakravarty

1 Terminologies

1. **State**
   A state is the minimal set of variables (position, velocity, orientation, etc.) that completely describe the system at a given time. It's the robot's full memory card snapshot — enough to predict the future with the system dynamics.

2. **Control Input**
   The command you give the robot: forces, torques, wheel speeds — anything that causes it to change state. Denoted $u(t)$ in continuous time or u_k in discrete time.

3. **Dynamics**
   The mathematical rules that govern how the state evolves in time: $\dot{x} = f(x, u)$ or $x_{k+1} = f(x_k, u_k)$. They tell you: "If I'm here and I do this, where will I be next?"

4. **Double Integrator Model**
   A basic physical model: acceleration is the control input, velocity is its integral, and position is the integral of that. Classic example: $\ddot{x} = u$. Good for modeling motion when there are no nasty nonlinearities.

5. **Kinematic vs Dynamic Models**
   Kinematic models assume perfect control of velocity and ignore forces — great for low-speed or planning. Dynamic models consider mass, inertia, and Newton's laws — required when physics matters (e.g., drones, rockets, race cars).

6. **Linear System**
   A system where dynamics follow the form: $x_{k+1} = Ax_k + Bu_k$. Simple, elegant, and heavily studied — a sweet spot for control theory and optimal control.

7. **Affine System**
   Just a linear system with a constant bias: $x_{k+1} = Ax_k + Bu_k + c$. Still manageable in most control setups and often shows up in real applications.

8. **Nonlinear System**
   A system where the dynamics don't follow a linear form — the wild ones. They can be hard to control but are often closer to how real robots behave.

9. **Discrete-Time System**
   Systems where state updates happen at regular time steps: $x_{k+1} = f(x_k, u_k)$. Like flipping through the robot's life one frame at a time.

10. **Continuous-Time System**
    Here, time flows smoothly: $\dot{x} = f(x, u)$. Great for theoretical analysis and modeling physical systems.

11. **Feedback Control**
    A strategy where the controller uses current state measurements to compute the input: $u = K(x - x_{\text{ref}})$. Like adjusting your bike's steering in real time to stay on path.

12. **Open-Loop Control**
    You precompute the control sequence without reacting to how the system behaves. Works only when everything goes exactly as planned — which is rare!

13. **Stability**
    A system is stable if it resists disturbances — it doesn't go haywire when nudged. Lyapunov stability ensures that small deviations don't grow over time.

14. **Finite Horizon**
    Optimization over a fixed number of future steps, like in Model Predictive Control (MPC). Instead of planning forever, you just focus on the next N seconds.

15. **Cost Function**
    Quantifies how "bad" a trajectory is. We try to minimize it. Typically: $J = \sum_{k=0}^{N}(x_k^T Q x_k + u_k^T R u_k)$. It's like a score — lower is better.

16. **Constraint**
    Limits on states or controls — e.g., max speed, joint angles, obstacle avoidance. Constraints make the problem realistic but also trickier.

17. **Chance Constraint**
    A probabilistic version of a constraint. Instead of "always stay within bounds," we say: "stay within bounds 95% of the time." Useful when noise and uncertainty are unavoidable.

18. **Trajectory**
    A path through state space over time. Could be smooth and continuous or made up of discrete waypoints. Think of it as the robot's plan for moving from A to B.

19. **Optimal Control**
    Finds the best control inputs to minimize a cost function while satisfying dynamics and constraints. Like solving a maze while saving energy and avoiding walls.

20. **Model Predictive Control (MPC)**
    A control method that solves an optimization problem at each time step using a finite horizon. You re-plan every few milliseconds — like a chess master calculating best moves on the fly.

21. **Bang-Bang Control**
    A type of control where inputs switch instantly between extremes. Think: gas pedal fully on or fully off — no in-between.

22. **Saturation**
    The limit of how much control you can apply — e.g., max torque. Controllers must respect these bounds to avoid unrealistic commands.

23. **Disturbance**
    An external force or effect that the controller didn't expect — like wind hitting a drone. Good controllers compensate for it.

24. **Process Noise**
    Uncertainty in how the system behaves — like a wheel slipping even though you told it to rotate precisely.

25. **Measurement Noise**
    Errors in sensing the state. Sensors lie (a bit), and your controller has to make decisions based on these slightly flawed measurements.

26. **Adaptive Control**
    A controller that adjusts its parameters as it learns more about the system — great for handling unknown or changing dynamics.

27. **Reinforcement Learning (RL)**
    A trial-and-error based learning framework where the robot learns a policy by maximizing rewards. Think of it as the robot figuring things out through experience.

2 Open-Loop vs Closed-Loop Control

Open-Loop vs Closed-Loop Control

Open-loop and closed-loop controls are the two fundamental approaches used in control systems. They represent two very different methods of controlling a system, each with its own advantages and disadvantages.

Open-loop control refers to systems that operate without feedback. In these systems, the control input is applied based on a set of predefined conditions or instructions, without any measurement or correction based on the system's output. The controller has no knowledge of the output of the system, and thus the performance of the system is assumed to behave according to the model or expectations.

Open-Loop Control

In an open-loop system, the input to the system is set in advance, and the system operates without any corrective actions based on the output. It is a "set-it-and-forget-it" control method, often used when the process is predictable, and disturbances or changes in system behavior are minimal. For example, in a simple water heater, the temperature is set using a thermostat, and the heater operates at that setting without measuring the actual temperature.

Key characteristics:

- No feedback or correction based on output.
- Simplicity and lower cost.
- Effective when system behavior is well understood and disturbances are minimal.
- Vulnerable to errors from disturbances, external forces, or system changes.

Closed-Loop Control

In contrast, closed-loop control systems use feedback to adjust the input based on the actual output. The system continuously monitors its output, compares it to the desired setpoint, and adjusts the input accordingly to correct any errors. This feedback loop ensures that the system maintains stability and accuracy despite disturbances or changes in system dynamics. For example, in a cruise control system, the car continuously measures its speed (output) and adjusts the throttle (input) to maintain a set speed.

Key characteristics:

- Feedback is used to adjust inputs based on the output.

- More accurate and robust in handling disturbances and model uncertainties.
- Can be more complex and costly due to the need for sensors, controllers, and feedback loops.
- Essential for systems requiring precise control and stability in changing environments.

The main advantage of closed-loop control is its ability to adapt to external changes and disturbances, while open-loop control is simpler but less robust in dynamic environments.

3 System Dynamics

System Dynamics

Understanding how systems evolve over time is a critical aspect of control theory. System dynamics refers to the mathematical description of how the state of a system changes over time in response to inputs. This is typically modeled using differential equations or difference equations.

In control systems, the system dynamics describe how the internal state of a system evolves based on inputs, initial conditions, and its governing laws. The system's state is often represented by a vector $x(t)$, and the input by a vector $u(t)$.

State-Space Representation

A system's dynamics can be represented in state-space form, where the system is described by a set of first-order differential equations. The state-space representation is crucial for modern control techniques such as state feedback and optimal control.

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

Where:

- $x(t)$ is the state vector of the system, representing all the variables that define the system's current condition.
- $u(t)$ is the control input, affecting the system's state.
- A is the state matrix, defining how the current state affects its rate of change.
- B is the input matrix, which determines how inputs influence the state.
- C and D are the output matrices that map the state and input to the system's output.

State-space models can handle multi-input multi-output (MIMO) systems and provide a clear picture of how the system behaves over time, given initial conditions and inputs.

Differential Equations

The state-space model is typically derived from physical laws such as Newton's laws for mechanical systems or Kirchhoff's laws for electrical circuits. These laws lead to differential equations that describe how the system's state evolves. For example, a simple mass-spring-damper system can be modeled by the following second-order differential equation:

$$m\ddot{x}(t) + c\dot{x}(t) + kx(t) = u(t)$$

Where:

- m is the mass of the object.
- c is the damping coefficient.
- k is the spring constant.
- $x(t)$ is the displacement of the mass.
- $u(t)$ is the external force acting on the mass.

This second-order equation can be converted to a first-order state-space form for control purposes.

4 Transfer Functions

Transfer Functions

Transfer functions are a powerful tool in control theory, used primarily for analyzing and designing linear time-invariant (LTI) systems in the frequency domain. They provide a relationship between the input and output of a system, making them especially useful in system analysis and controller design.

The transfer function of a system is defined as the ratio of the Laplace transform of the output to the Laplace transform of the input, assuming initial conditions are zero. It is typically denoted as $G(s)$, where s is the complex frequency variable in the Laplace transform.

Mathematical Definition

The transfer function is given by:

$$G(s) = \frac{Y(s)}{U(s)}$$

Where:

- $G(s)$ is the transfer function of the system.
- $Y(s)$ is the Laplace transform of the output.
- $U(s)$ is the Laplace transform of the input.

Transfer functions are useful for determining the system's behavior in the frequency domain, which is important for analyzing stability, resonance, and response to various inputs.

Applications and Use Cases

Transfer functions are widely used in the analysis and design of control systems for the following purposes:

- **Stability Analysis**: By examining the poles and zeros of the transfer function, one can assess the stability of the system. For example, if all poles have negative real parts, the system is stable.

- **Frequency Response**: Transfer functions allow engineers to analyze how a system reacts to sinusoidal inputs at various frequencies. This is used to design controllers (e.g., PID controllers) that ensure desired system performance.
- **Controller Design**: The transfer function is a key component in designing controllers, such as PID, that ensure the system meets performance criteria (like settling time, overshoot, etc.).

Example: Transfer Function of a Mass-Spring-Damper System

For a mass-spring-damper system described by:

$$m\ddot{x}(t) + c\dot{x}(t) + kx(t) = u(t)$$

The Laplace transform yields the transfer function:

$$G(s) = \frac{1}{ms^2 + cs + k}$$

This transfer function describes how the displacement of the mass responds to an applied force $u(t)$ in the frequency domain.

Limitations of Transfer Functions

While transfer functions are useful, they are typically limited to linear, time-invariant systems and do not provide a comprehensive model for nonlinear or time-varying systems. More advanced techniques, such as state-space modeling, are required to handle such systems.

5 Bang-Bang Control

Vibes

All or nothing. Full send or full brake. Used where simplicity wins over finesse.

Control Law

$$u(t) = \begin{cases} +U, & e(t) > 0 \\ -U, & e(t) < 0 \end{cases}$$

Key Idea

No middle ground. The controller acts like an aggressive binary switch. Works well when you don't care about being smooth—just reach the goal ASAP.

Real-World Example

Thermostat: Heater turns ON at 18°C and OFF at 22°C. There's no "slightly warm" mode—it's either cooking or chilling.

Pros and Cons

- **Pros:** Simple to implement. No tuning.
- **Cons:** Can cause high wear/tear, oscillations, and overshoot.

6 PID Control (Proportional-Integral-Derivative)

Quick Vibe

This is the GOAT of control systems. It's used everywhere—from thermostats to drones. It's how you keep your robot from overshooting like a drunk driver.

Control Law

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de(t)}{dt}$$

Where:

- $e(t) = r(t) - y(t)$: Error between reference and actual output
- K_p: Proportional gain – how aggressively we react to current error
- K_i: Integral gain – how much we care about past accumulated error
- K_d: Derivative gain – how much we predict future error

Tuning Intuition

- **Too much** K_p: Fast response, but can overshoot and oscillate
- **Too much** K_i: Removes steady-state error, but may cause instability
- **Too much** K_d: Dampens oscillations, but sensitive to noise

Real-World Example

Cruise Control in a Car: You want to go 100 km/h. - K_p adjusts throttle based on how far off you are - K_i slowly adds pressure if you're consistently under-speed - K_d backs off throttle if it senses you're speeding up too fast

7 Path Following Control

Path Following

Path following is a control strategy designed to make a system or robot follow a predefined path as closely as possible, often with real-time adjustments.

What It Does

Path following is a technique used to ensure that a vehicle or robot moves along a predefined path, compensating for errors or disturbances that could cause deviation. It is crucial for applications such as autonomous vehicles, drones, and robotic systems where precise motion along a desired trajectory is required.

Mathematical Formulation

Path following control typically involves minimizing the distance between the system's current position and the desired path, often modeled using errors in the lateral and longitudinal directions. The control law can be expressed as:

$$u(t) = -K_e \cdot e(t)$$

Where: - $e(t)$ is the error vector (e.g., lateral and angular errors), - K_e is the feedback gain matrix.

8 State-Space Representation

Why You Should Care

State-space is the control engineer's language. Whether it's LQR, MPC, or Kalman Filter, this is where it all begins.

General Form

$$\dot{x}(t) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t) + Du(t)$$

- $x(t) \in \mathbb{R}^n$: state vector (e.g., position, velocity, temperature, etc.)
- $u(t) \in \mathbb{R}^m$: control input (forces, torques, voltages)
- $y(t) \in \mathbb{R}^p$: measured outputs

Interpretation

Think of it like tracking a robot's life story using math. - The A matrix tells how your current state affects the future. - The B matrix tells how the input pushes the state. - The C matrix picks out what you're observing. - The D matrix (often 0) captures direct feedthrough from input to output.

Example: Inverted Pendulum on a Cart

Let:

$$x(t) = \begin{bmatrix} \text{cart position} \\ \text{cart velocity} \\ \text{pole angle} \\ \text{pole angular velocity} \end{bmatrix}, \quad u(t) = \text{force applied on cart}$$

Then define system using physics (Newton/Euler) to get matrices A, B, etc.

Why Use This?

- You can design optimal controllers (like LQR)
- Simulate multi-variable systems
- Model sensor/actuator dynamics precisely

9 Lyapunov Stability

Lyapunov Stability

Lyapunov's method provides a way to analyze and ensure the stability of a system using mathematical functions. It's about proving that the system will not diverge.

What It Does

Lyapunov stability involves finding a scalar function (Lyapunov function) that shows the system's energy is dissipative, meaning the system will eventually settle into a stable state.

Mathematical Formulation

Lyapunov's method asserts that if a Lyapunov function $V(x)$ satisfies:

$$\dot{V}(x) < 0, \quad \forall x \neq 0,$$

then the system is stable at the equilibrium point.

Real-World Intuition

It's like ensuring that a pendulum will always come back to rest at its lowest point after being disturbed, as long as no external force keeps it swinging.

Key Challenges

- **Finding a Lyapunov function**: Not always possible to find one for complex systems.
- **Inconclusive results**: Lyapunov's method provides stability proof but not necessarily the speed of convergence.

Why It's Great

- Provides a **rigorous stability criterion**.
- Useful for **nonlinear systems**.

10 Robust Control

Robust Control

Robust control ensures that the system behaves as expected despite uncertainties and disturbances. It's about making the system resilient to external factors.

What It Does

Robust control is used for systems that face model uncertainties, external disturbances, or unpredictable environments. The controller is designed to ensure stability and performance even under these adverse conditions.

Mathematical Formulation

Robust control typically involves designing a controller that minimizes the worst-case scenario of system performance, often using the *H-infinity* norm:

$$\min_{\{u_k\}} \|G(s)\|_\infty \quad \text{subject to} \quad G(s) \in \mathcal{G}$$

Where: - $G(s)$ is the transfer function of the system, - $\mathcal{G}$ is the set of uncertainties.

Real-World Intuition

Think of a fighter jet controller that ensures the plane remains stable and responsive even when subjected to sudden wind gusts or mechanical issues.

Key Challenges

- **Over-conservatism**: The controller may be too conservative, affecting performance.
- **High complexity**: Designing robust controllers can be computationally demanding.

Why It's Great

- Ensures **system stability** despite uncertainties.
- Provides **strong performance guarantees** in unpredictable conditions.

11 Pure Pursuit and Stanley Control

Pure Pursuit + Stanley

Pure Pursuit and Stanley are like two superhero algorithms for autonomous vehicles, making them steer towards their goals in a fun yet highly effective way.

Pure Pursuit

Pure Pursuit is a simple, intuitive algorithm for path tracking. It works by looking ahead at a fixed distance along the path and steering towards that point. Think of it like walking towards the point where a friend is pointing, but always adjusting to maintain a steady direction.

Key Idea

- Pick a point on the trajectory (ahead of the vehicle) - Compute the steering angle that makes the vehicle go towards that point

Mathematical Formula

The steering angle δ is computed using the geometric relationship:

$$\delta = \arctan\left(\frac{2L\sin(\alpha)}{d}\right)$$

Where: - L is the wheelbase of the vehicle - α is the angle between the vehicle's heading and the line to the look-ahead point - d is the distance to the look-ahead point

Steps to Implement Pure Pursuit

1. Determine the look-ahead distance d, which is a function of speed: $d = k \cdot v$, where k is a constant and v is the current velocity. 2. Find the look-ahead point. 3. Calculate the steering angle δ using the formula. 4. Apply the steering angle in control.

Real-World Intuition

Imagine driving a car and trying to keep the car on a curved road. Instead of following the road immediately ahead of you, you focus on a point some distance ahead. As you move towards that point, you adjust your steering to keep the car aimed at that point.

Pros and Cons

- **Pros:** Easy to implement, real-time, intuitive.
- **Cons:** May overshoot or oscillate if the look-ahead distance is too large or small.

Stanley Controller

Stanley is an extension of Pure Pursuit, and it combines path following with feedback on the vehicle's orientation to improve stability. It's like Pure Pursuit with a built-in corrective mechanism to prevent drifting.

Key Idea

- It minimizes the cross-track error (how far the vehicle is off from the desired path). - It also corrects for the vehicle's heading error (the difference between the vehicle's orientation and the direction of the path).

Mathematical Formula

The Stanley controller uses the following equation to calculate the steering angle δ:

$$\delta = \theta + \arctan\left(\frac{k \cdot e_{\text{ct}}}{v}\right)$$

Where: - θ is the vehicle's heading angle - e_{ct} is the cross-track error - k is a gain parameter (determines how aggressively the controller corrects) - v is the velocity of the vehicle

Steps to Implement Stanley

1. Compute the cross-track error e_{ct} as the perpendicular distance from the vehicle to the closest point on the path. 2. Compute the heading error θ, the difference between the vehicle's orientation and the path direction. 3. Apply the correction term to the steering angle.

Real-World Intuition

Imagine your car is veering slightly off the road. Stanley doesn't just tell you where to go; it also tells you how to correct your heading based on how far off track you are, like your steering wheel vibrating to tell you when to correct. It's especially helpful when the path is noisy or curved!

Pros and Cons

- **Pros:** More stable than Pure Pursuit, good for noisy paths.
- **Cons:** Can be sensitive to tuning of the gain k, requires computing cross-track error.

12 Sliding Model Control

Sliding Mode Control (SMC)

SMC is a control method that forces the system state to "slide" along a predetermined surface, ensuring fast and precise control.

What It Does

Sliding Mode Control forces the system state to remain on a specific surface (the sliding surface) by designing a control law that switches between different modes, ensuring robustness against disturbances and uncertainties.

Mathematical Formulation

The sliding surface $s(x)$ is typically chosen as:

$$s(x) = Cx + D \quad \text{and the control law is} \quad u = -Ks(x),$$

where C and D are constants, and K is a gain to ensure the system remains on the surface.

Real-World Intuition

It's like driving a car that always adjusts its steering and throttle to stay exactly on a straight road despite bumps or curves in the path.

Key Challenges

- **Chattering**: Frequent switching of control modes can lead to high-frequency oscillations.
- **Tuning the sliding surface**: Choosing the correct sliding surface is crucial for performance.

Why It's Great

- Highly **robust** to disturbances and model uncertainties.
- Provides precise control, especially in the presence of **nonlinearities**.

13 Adaptive Control

Adaptive Control

Adaptive control is all about adjusting control parameters in real-time to cope with unknown or changing system dynamics. It's like having a system that learns to adapt to new conditions as they evolve.

What It Does

Adaptive control is designed for systems whose parameters are uncertain or time-varying. It modifies the controller based on real-time feedback, allowing it to adjust to changing dynamics or external disturbances.

Mathematical Formulation

At each time step, adaptive control typically modifies control laws through an update mechanism based on error feedback. The control law can be expressed as:

$$u(t) = K(t)x(t) + \Delta u(t)$$

Where: - $u(t)$ is the control input, - $K(t)$ is the adaptive gain, - $\Delta u(t)$ is the correction to the control law.

Real-World Intuition

Imagine a robot that adjusts its movement strategy as it encounters unexpected obstacles or terrain changes, ensuring its motion remains optimal despite environmental changes.

Key Challenges

- **Slow adaptation**: Adjusting to new dynamics can take time.
- **Stability concerns**: Ensuring the system remains stable as the controller adapts.

Why It's Great

- Works well with **uncertain or time-varying systems**.
- Provides **real-time adjustments** based on system feedback.

14 Kalman Filter (KF)

Nerd Vibes

Kalman Filter is your mathy best friend who helps you guess the unobservable world using sensors and a bit of physics.

System Model (Linear + Gaussian)

$$x_{k+1} = Ax_k + Bu_k + w_k, \quad w_k \sim \mathcal{N}(0, Q) y_k = Cx_k + v_k, \quad v_k \sim \mathcal{N}(0, R)$$

- x_k: true (hidden) state - y_k: noisy measurement - w_k, v_k: process and measurement noise

Two Main Steps

Prediction Step:

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1} + Bu_{k-1} P_{k|k-1} = AP_{k-1|k-1}A^\top + Q$$

Update Step:

$$K_k = P_{k|k-1}C^\top (CP_{k|k-1}C^\top + R)^{-1} \hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(y_k - C\hat{x}_{k|k-1}) P_{k|k} = (I - K_kC)P_{k|k-1}$$

Real-World Example

GPS on a drone: - Prediction: Based on last known velocity and position - Update: Fuse new GPS reading (which may be noisy) This lets you estimate where the drone really is—even with crappy GPS!

15 Control Lyapunov Functions

Control Lyapunov Functions (CLF)

Control Lyapunov Functions are designed to make control laws directly influence the stability of a system, creating efficient stability guarantees for nonlinear systems.

What It Does

CLF integrates control and stability analysis, where the Lyapunov function itself is used to design control laws that guarantee system stability.

Mathematical Formulation

A typical CLF approach involves solving an optimization problem to find a control law that minimizes the Lyapunov function, ensuring stability at each step:

$$\min_{\{u_k\}} V(x) \quad \text{subject to} \quad \dot{V}(x) < 0$$

Where $V(x)$ is a Lyapunov function.

Real-World Intuition

Imagine designing a control system where the controller itself adapts in real-time to ensure that the system stays in a stable operating region, like a drone maintaining a steady hover in turbulent wind conditions.

Key Challenges

- **Computationally intensive**: Solving optimization problems at every time step.
- **Requires accurate system models** for effective design.

Why It's Great

- Provides **guaranteed stability** for nonlinear systems.
- Useful for **complex systems** where traditional methods struggle.

16 Control Barrier Functions

Control Barrier Functions (CBF)

Control Barrier Functions ensure that a system operates within a safe region, preventing it from violating safety constraints.

What It Does

CBFs provide a way to enforce safety constraints by ensuring that the system's state remains within a predefined safe region, even in the presence of disturbances or uncertainties.

Mathematical Formulation

The CBF ensures safety by ensuring that the time derivative of a safety function $h(x)$ is always non-positive:

$$\dot{h}(x) \geq -\alpha(h(x)),$$

where α is a function designed to enforce safety.

Real-World Intuition

Think of a robot that is required to stay within a designated safe area, with its controller ensuring it never crosses boundaries, even when unexpected obstacles appear.

Key Challenges

- **Conservative safety margins**: The system may be overly restricted to stay within safe regions.
- **Safety vs. performance trade-off**: Balancing safety with task performance can be challenging.

Why It's Great

- Provides **guaranteed safety** for dynamic systems.
- Ensures that the system remains within **safe operational boundaries**.

Real-World Intuition

Imagine a drone that follows a flight path, adjusting its speed and direction as it goes to stay exactly on the trajectory despite external factors like wind or obstacles.

Key Challenges

- **Environmental disturbances**: Factors like wind, rough terrain, or obstacles can make path following challenging.
- **Precise error measurement**: Accurately measuring the path error is vital for performance.

Why It's Great

- Ensures **high precision** in moving along predefined paths.
- Useful for **autonomous systems** that require real-time trajectory adjustments.
- Can be applied in various fields like **robotics, automotive, and aerospace**.

17 Linear Matrix Inequalities (LMI)

LMI

Linear Matrix Inequalities (LMI) provide a framework for solving complex control problems, particularly for ensuring system stability and performance in a structured, efficient way.

What It Does

LMIs are a mathematical tool used to formulate and solve optimization problems in control theory. They allow for the design of controllers that ensure system stability and performance, particularly in the presence of uncertainties or constraints.

Mathematical Formulation

An LMI is a constraint that can be expressed as:

$$A(x) = \sum_i A_i x_i \quad \text{such that} \quad A(x) \prec 0$$

Where: - A_i are matrices and x_i are variables, - The inequality $\prec 0$ implies that the matrix is negative definite.

Real-World Intuition

LMIs can be thought of as tools to design a system controller that guarantees stability and performance while keeping the system within a set of constraints. For instance, it's like designing a building's structure where you optimize the materials and forces to ensure it remains stable under varying conditions.

Key Challenges

- **High computational cost**: Solving LMIs can be computationally demanding.
- **Complex formulation**: Defining appropriate LMIs for complex systems can be challenging.

Why It's Great

- Provides a **systematic framework** for control design.
- Guarantees **robust stability and performance**.
- Can handle **uncertainties and constraints**.

18 Linear Quadratic Regulator (LQR)

Intuition Vibes

LQR is like being the chill but efficient controller—you don't freak out like a bang-bang guy or over-obsess like PID. You're minimizing "energy" to balance precision and effort.

Problem Setup

Given a linear system:

$$\dot{x}(t) = Ax(t) + Bu(t)$$

You want to minimize the cost:

$$J = \int_0^\infty \Big(x(t)^\top Q x(t) + u(t)^\top R u(t) \Big)\, dt$$

- $Q \succeq 0$: Penalizes bad states (e.g., being off track) - $R \succ 0$: Penalizes large inputs (e.g., wasting energy)

Control Law

$$u(t) = -Kx(t), \quad \text{where } K = R^{-1}B^\top P$$

P is the solution to the **Continuous Algebraic Riccati Equation (CARE)**:

$$A^\top P + PA - PBR^{-1}B^\top P + Q = 0$$

Real-World Analogy

You're balancing a stick on your palm. LQR ensures you don't wave your arm wildly (minimize effort) and you keep the stick upright (minimize error).

19 Model Predictive Control (MPC)

Vibes

MPC is like playing chess with your robot. You think multiple moves ahead and pick the best one—at every time step.

Basic Idea

At every time step:

1. Predict the system's behavior over a future horizon N using dynamics
2. Solve an optimization problem to minimize a cost (like in LQR)
3. Apply the first control input u_0 only
4. Re-plan at the next time step (feedback)

Cost Function

$$J = \sum_{k=0}^{N-1} \Big(x_k^\top Q x_k + u_k^\top R u_k \Big) + x_N^\top Q_f x_N$$

Subject to Constraints

$$x_{k+1} = Ax_k + Bu_k$$
$$x_k \in \mathcal{X}, \quad u_k \in \mathcal{U}$$

- Can handle state and control bounds (e.g., max speed, torque limits) - Can add obstacle avoidance constraints too!

Real-World Analogy

Self-driving car approaching a turn: - MPC checks possible paths and speeds - Chooses one that gets you through safely and smoothly - Recalculates every 0.1s as road/traffic changes

Why It Rocks

- Can handle constraints explicitly
- Future-aware: takes context into account
- Naturally works with LQR ideas for the terminal cost

20 Nonlinear Model Predictive Control (NMPC)

Nonlinear MPC

Nonlinear MPC is the granddaddy of optimal control, dealing with **nonlinear dynamics** and **constraints**. It's like going beyond the basics to make real-time decisions with an understanding of your future behavior.

What It Does

NMPC is an advanced method for controlling systems with nonlinear dynamics. Unlike LQR, which works best for linear systems, NMPC optimizes **over a finite time horizon** and can handle constraints like obstacles.

Mathematical Formulation

At each time step, NMPC solves the following optimization problem:

$$\min_{\{u_k\}} \sum_{k=0}^{N-1} \ell(x_k, u_k) + \phi(x_N)$$

Subject to:

$$x_{k+1} = f(x_k, u_k), \quad x_0 = x_{\text{initial}}, \quad x_k \in \mathcal{X}, \quad u_k \in \mathcal{U}$$

Where: - x_k is the state at time k - u_k is the control input at time k - $f(x_k, u_k)$ is the nonlinear system dynamics

Real-World Intuition

NMPC is like having a self-driving car that not only plans its path but **constantly re-plans** based on where it is, while factoring in obstacles, traffic, and other constraints.

Key Challenges

- **Computationally intensive**: Solving the optimization problem in real-time can be tricky.
- **Handling constraints**: Particularly when obstacles and other dynamic constraints are involved.

Why It's Great

- Handles **highly nonlinear dynamics**.
- Can handle **hard constraints** (like obstacles or physical limits).
- **Adaptive**: Re-plans based on new information in real-time.

21 Sampling-Based Control: MPPI (Model Predictive Path Integral)

Cool Nerd Talk

MPPI is what happens when you merge path integrals from physics with control—like Monte Carlo LQR with bonus stochastic flavor.

Big Picture

Instead of solving an optimization problem with gradients, MPPI samples a bunch of noisy control trajectories and picks the best one (in expectation).

Used when:

- The system is nonlinear and stochastic
- Gradient-based optimization is hard or unstable

Dynamics

$$x_{k+1} = f(x_k, u_k) + \mathcal{N}(0, \Sigma)$$

Cost of a Trajectory

For each sampled control sequence $\{u_0^i, \ldots, u_{T-1}^i\}$, compute the trajectory cost:

$$S^i = \sum_{k=0}^{T-1} q(x_k^i, u_k^i) + \phi(x_T^i)$$

Control Update

Use path integral to compute weighted average:

$$u_k = \sum_{i=1}^{N} \omega^i u_k^i, \quad \omega^i = \frac{\exp\left(-\frac{1}{\lambda} S^i\right)}{\sum_j \exp\left(-\frac{1}{\lambda} S^j\right)}$$

- λ is the temperature: higher = more exploration, lower = greedier

Intuition

More optimal trajectories get higher weights. The idea is similar to **importance sampling** in statistics.

Real-World Example

Imagine you simulate 1000 slightly noisy ways to ride your skateboard down a slope. MPPI says: "Let's average the best ones, but weighted toward the safest/smoothest."

Why It's Dope

- Handles highly nonlinear dynamics
- No need to compute gradients
- Parallelizable with GPUs!

22 Trajectory Optimization (Direct Methods)

Deep Nerd Vibes

Trajectory optimization is like choreographing a dance—solve for a full trajectory ahead of time by discretizing space-time and optimizing it.

Direct Collocation (Most Popular)

You discretize time into N intervals and treat states x_k and controls u_k as decision variables.

Optimization Form

$$\min_{\{x_k, u_k\}} \sum_{k=0}^{N-1} \ell(x_k, u_k) + \phi(x_N)$$

Subject to:

$$x_{k+1} = f(x_k, u_k) \quad \text{(system dynamics)} x_k \in \mathcal{X}, \quad u_k \in \mathcal{U}$$

Collocation Constraint

You enforce the system dynamics using numerical integration schemes like:

$$x_{k+1} = x_k + hf(x_k, u_k) \quad \text{(Euler)} \text{or use midpoint, trapezoid, Runge-Kutta, etc.}$$

Why It's Hard

- It's a large nonlinear programming (NLP) problem
- You need good solvers (like IPOPT, SNOPT, etc.)

Real-World Example

Want a humanoid robot to jump over a block? You can't "react" to that—you need to optimize the jump trajectory offline, then execute it.

Extras

- Add slack variables to soften constraints - Use "knot points" for coarse-to-fine refinement - Warm-start with a feasible initial guess

Why It's Core

- Foundation for acrobatics and offline optimal motions
- Bridge to tools like DDP, iLQR, and NLP-based planners

23 Differential Dynamic Programming (DDP) and Iterative LQR (iLQR)

DDP + iLQR

Differential Dynamic Programming (DDP) and Iterative LQR (iLQR) are like the top-tier strategies when you need **optimal control** in nonlinear systems. They offer ways to plan trajectories that **minimize cost** while accounting for dynamics.

Differential Dynamic Programming (DDP)

DDP is like LQR but takes it to the next level by handling nonlinear dynamics. It's used to solve optimal control problems by iterating to refine the trajectory.

Key Idea

- Linearize the dynamics around the current trajectory. - Solve the resulting **linearized** problem using a feedback control law, and update the trajectory.

Mathematical Form

The objective is to minimize the cost function:

$$J = \sum_{k=0}^{N-1} \left(x_k^\top Q x_k + u_k^\top R u_k \right) + \phi(x_N)$$

Where: - x_k are states at time k - u_k are control inputs at time k - Q, R are weight matrices for states and controls - $\phi(x_N)$ is the terminal cost

DDP iterates to refine the trajectory and control law by linearizing the system dynamics:

$$x_{k+1} = f(x_k, u_k) + \delta x_k$$

Real-World Intuition

Imagine trying to play a perfect game of darts. You throw one dart, and DDP tells you where to throw the next dart based on the current result—each dart throw is a feedback from the previous one to get closer to the bullseye.

iLQR: A Simplified Version of DDP

iLQR is a simplified version that uses **local linearization** and works with **fixed horizon** problems. It's like a baby cousin to DDP but faster and easier to implement.

Key Idea

- Use linearization around the trajectory for each iteration. - Use feedback control at each step, iterating to improve the trajectory.

Real-World Intuition

Think of it as steering a car around a bend: iLQR helps you plan the path in small steps while correcting your steering at each step until you smoothly round the bend.

Why It's Cool

- Can solve **complex** nonlinear control problems.
- **Efficient** compared to brute-force optimization methods.
- Widely used for **robotic motion** and **autonomous vehicles**.

24 Covariance Steering (CS)

Vibes

Covariance Steering is like planning not only where you want to be, but how confident (uncertain) you want to be when you get there. It's trajectory + uncertainty shaping.

What It Does

You specify both:

- A desired final mean state: μ_T
- A desired final covariance: Σ_T

This is especially useful when working with **uncertain systems** or when you want to shape the distribution of a robot's belief, not just its expected location.

Problem Setup

Linear stochastic dynamics:

$$x_{k+1} = A_k x_k + B_k u_k + w_k, \quad w_k \sim \mathcal{N}(0, W_k)$$

We want:

$$x_0 \sim \mathcal{N}(\mu_0, \Sigma_0), \quad x_T \sim \mathcal{N}(\mu_T, \Sigma_T)$$

Design u_k to: - steer the **mean** trajectory - control the **dispersion** (covariance)

Mathematical Flavor

Split control into:

$$u_k = v_k + K_k(x_k - \mu_k)$$

- v_k: open-loop to move the mean - K_k: feedback to shape the uncertainty

The evolution of the **mean** follows:

$$\mu_{k+1} = A_k \mu_k + B_k v_k$$

The **covariance** evolves as:

$$\Sigma_{k+1} = (A_k + B_k K_k)\Sigma_k (A_k + B_k K_k)^\top + W_k$$

Real-World Vibes

Say you're piloting a drone through wind. You don't just want to reach a window—you want to squeeze uncertainty so it fits through it!

Why It's Powerful

- Controls belief—not just the state!
- Forms the basis of modern safe stochastic motion planning

25 Stochastic Control

Stochastic Control

Stochastic control is a method that deals with systems affected by random processes, making it essential for systems where uncertainty plays a significant role.

What It Does

Stochastic control is used to design control strategies for systems influenced by randomness or noise. These strategies are optimized to minimize the effects of uncertainty and make systems robust to random disturbances.

Mathematical Formulation

Stochastic control problems typically involve minimizing a cost function that accounts for both the system dynamics and the uncertainty. The problem can be formulated as:

$$\min_{\{u_k\}} \mathbb{E}\left[\sum_{k=0}^{N-1} \ell(x_k, u_k) + \phi(x_N)\right]$$

Where: - $\mathbb{E}$ denotes the expected value operator, - x_k is the state, and u_k is the control input at time k.

Real-World Intuition

Stochastic control is like planning a route for an autonomous vehicle, where the vehicle's movement is influenced by unpredictable factors like traffic, road conditions, or weather. The controller makes decisions that are robust to these uncertainties.

Key Challenges

- **Complexity of randomness**: Modeling uncertainty and randomness accurately can be difficult.
- **Real-time optimization**: Solving stochastic optimization problems in real-time is computationally expensive.

Why It's Great

- Handles **systems with inherent randomness**.
- Provides **robust solutions** to unpredictable conditions.
- Widely applicable in areas like **finance, robotics, and aerospace**.

26 Energy-Based Control

Energy-Based Control

Energy-based control focuses on the principles of energy conservation and dissipation to design controllers that regulate the system's energy state over time.

What It Does

Energy-based control strategies aim to manipulate the system's energy, ensuring that it evolves toward a desired state while minimizing energy usage or maximizing efficiency. This is especially useful in mechanical systems, electrical circuits, and other energy-sensitive applications.

Mathematical Formulation

Energy-based control often relies on the system's energy function $E(x)$, and the control law is designed to ensure that the energy is minimized or maximized over time:

$$\dot{E}(x) = \frac{\partial E}{\partial x} \cdot f(x) + u^T g(x)$$

Where: - $E(x)$ is the energy function of the system, - $f(x)$ is the system dynamics, and u is the control input.

Real-World Intuition

Energy-based control is like designing a thermostat that adjusts the heating or cooling in a house to maintain a stable temperature while minimizing energy consumption, or a robot that optimizes its movements to conserve power.

Key Challenges

- **Energy modeling**: Accurately modeling the system's energy behavior can be complex.
- **Efficiency trade-offs**: Balancing energy conservation with other performance metrics can be tricky.

Why It's Great

- Ideal for **energy-efficient control systems**.
- Provides **direct control** over the system's energy state.
- Widely used in applications like **robotics, electrical systems, and renewable energy**.