Control System Engineering

Iheb BOUARICHE | Automatic control | 11-08-2023

Dynamic systems and control

Automatic control:

Automatic control, also known as control engineering or simply control systems, is a field of engineering that deals with designing, analyzing, and implementing systems to regulate and manipulate the behavior of dynamic systems. The goal of automatic control is to ensure that a system behaves in a desired manner, whether it's maintaining a specific temperature, guiding a vehicle, or controlling the speed of a motor.

Control and regulation of a system:

Control: In control engineering, it refers to the process of influencing the behavior of a system by manipulating its inputs or parameters. Control systems (or "control laws" covered later) are designed to steer a system's output or behavior towards a desired goal. This can involve maintaining stability, achieving a particular trajectory, or meeting performance specifications. Control strategies are developed to ensure that the system responds appropriately to changes in its environment or inputs.

Regulation: in control engineering specifically focuses on maintaining a system's output or state at a desired setpoint or reference value. It involves continuously adjusting the system's inputs or parameters to counteract any deviations from the desired state caused by disturbances or changes in operating conditions. The goal of regulation is to minimize the difference between the desired setpoint and the actual system output, ensuring that the system operates within specified limits.

Physical system:

A physical system is a collection of physical components or entities that interact with each other and follow certain rules or laws of physics. These components can include objects, particles, fields, forces, and various measurable properties. Physical systems can be as simple as a single object moving through space or as complex as a network of interconnected components in a biological organism of a human body or an engineered machine.

Dynamical system:

A dynamical system is a mathematical concept used to describe how a system changes over time. It's a way of modeling the evolution of a system's state or configuration as time progresses. A dynamical system consists of a set of equations or rules that govern how the system's state changes in response to its current state and external inputs. These equations can be deterministic (predictable) or stochastic (involving randomness). The state of a dynamical system is often represented by a set of variables, and the equations describe how these variables evolve over time.

Types of Dynamical System:

Dynamical systems are categorized based on their behavior and characteristics. Here are the different types:

1. Linear Dynamical System: A dynamical system is linear if it follows the principles of superposition and homogeneity. The system's response is directly proportional to its inputs.
2. Nonlinear Dynamical System: A dynamical system is nonlinear if it does not satisfy the principles of superposition and homogeneity. The relationship between inputs and outputs is not directly proportional.
   - **Principle of Superposition**: This principle states that the response of a linear system to a sum of multiple inputs is equal to the sum of the responses to each individual input acting alone. Mathematically, if $y_1(t)$ is the output of a system due to input $x_1(t)$ and $y_2(t)$ is the output due to input $x_2(t)$, then the output $y(t)$ due to the input $x(t) = x_1(t) + x_2(t)$ is given by $y(t) = y_1(t) + y_2(t)$.
   - **Principle of Homogeneity**: This principle states that the response of a linear system to a scaled input is equal to the same input scaled by the corresponding factor applied individually. Mathematically, if $y(t)$ is the output of a system due to input $x(t)$, then the output $y_a(t)$ due to input $a \cdot x(t)$ (where a is a constant) is given by $y_a(t) = a \cdot y(t)$.
3. Time-Invariant Dynamical System: A dynamical system is time-invariant if its behavior remains constant over time, regardless of when the inputs are applied.
4. Time-Varying Dynamical System: A dynamical system is time-varying if its behavior changes with time. The system parameters or characteristics may vary over time.
5. Continuous-Time Dynamical System: A dynamical system is continuous-time if its evolution is described by differential equations, and time is considered as a continuous variable.
6. Discrete-Time Dynamical System: A dynamical system is discrete-time if its evolution is described by difference equations, and time is considered in discrete steps.
7. Deterministic Dynamical System: A dynamical system is deterministic if its future behavior is completely determined by its current state and inputs.
8. Stochastic Dynamical System: A dynamical system is stochastic if it involves randomness or uncertainty, and future behavior is described in terms of probabilities.
9. Linear Time-Invariant (LTI) System: A combination of the linear and time-invariant types. The system follows both superposition and homogeneity principles and remains constant over time.

Linear System

"Single Input and Single Output LTI systems"

Dynamic linear system representation: dynamic system can be represented in various ways:

Temporal domain: In the temporal domain, the behavior of a physical system is described as a function of time. The system's response to various inputs and disturbances can be observed and analyzed over time. This representation often involves differential equations that describe how the system's state variables change with respect to time. We use time-domain analysis to study transient responses, stability, and time-based characteristics of the system.

Frequential domain: In the frequential domain, the behavior of a physical system is analyzed in terms of its frequency content. This is particularly useful when dealing with signals or systems that exhibit periodic or oscillatory behavior. The frequential domain representation involves converting time-domain signals or responses into frequency-domain representations using techniques like Fourier transforms. We use frequential domain analysis to understand how a system responds to different frequencies, including resonance phenomena and frequency-based characteristics, and also to study transient responses, stability, and time-based characteristics of the system.

Notes:

- We often switch between temporal and frequential domain to gain a comprehensive understanding of a system's dynamics and design appropriate control strategies.

- In the frequential domain, the mathematical representation of a system can often be simplified compared to the continuous-time domain, particularly for linear time-invariant (LTI) systems. This simplification occurs because complex differential equations in the time domain can be transformed into algebraic equations in the frequential domain. This transformation is facilitated by tools like the Laplace transform or the Fourier transform.

1) **Continuous representation:** used to model systems that change continuously in time. We have three types of continuous representation:

- **Differential Equation Representation:** is a mathematical expression that describes the relationship between the system's input, output, and internal states in terms of derivatives. It captures the system's behavior over time. For example, a first order system can be represented by the following differential equation:

$$\frac{dy(t)}{dt} = -a \cdot y(t) + b \cdot u(t)$$

In this equation:

- y(t) represents the system's output at time t.
- u(t) represents the system's input at time t.
- a and b are constants that determine the behavior of the system.

- **Transfer Function Representation**: transfer function is a mathematical representation used in control system engineering to describe the relationship between the input and output of a linear time-invariant (LTI) system in the frequential domain. It simplifies the analysis and design of control systems by representing the system's behavior as a ratio of polynomials in a complex variable, typically denoted as "s" in the Laplace domain.

The transfer function is defined as the Laplace transform of the system's output divided by the Laplace transform of its input, assuming zero initial conditions. Mathematically, for a continuous-time system, the transfer function $H(s)$ is given by:

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

Where:

- $Y(s)$ is the Laplace transform of the output response.
- $U(s)$ is the Laplace transform of the input signal.

- **State-Space Representation:** is a mathematical framework used in control system engineering to describe the behavior of dynamic systems. It provides a concise and flexible way to model and analyze system dynamics using a set of first-order ordinary differential equations.

In state-space representation, a dynamic system is described by two sets of equations:

State Equations: $\dot{x}(t) = Ax(t) + Bu(t)$ This equation represents how the system's state variables $x(t)$ change over time. The term $Ax(t)$ captures the system's internal dynamics, and $Bu(t)$ represents how the control input $u(t)$ influences the state variables. The matrix A determines the system's dynamic behavior, while B relates the control input to the state changes.

Output Equation: $y(t) = Cx(t) + Du(t)$ this equation relates the system's output $y(t)$ to its state variables and input. The term $Cx(t)$ indicates how the state variables contribute to the output, and $Du(t)$ represents the direct influence of the control input on the output. The matrix C determines the relationship between state variables and the output, while D accounts for the direct feedthrough of the input to the output.

In these equations:

- $x(t)$ represents the state vector, which contains the internal variables describing the system's state.
- $u(t)$ represents the control input vector.
- $y(t)$ represents the output vector.
- $A, B, C,$ and D are matrices that characterize the system's dynamics and relationships.

State-space representation offers several advantages:

- ✓ It can handle systems with multiple inputs and outputs.
- ✓ It can handle the internal state variation of the system.
- ✓ It accommodates both continuous-time and discrete-time systems.
- ✓ It provides a unified framework for analysis, modeling, and control design.

✓ It is well-suited for modern control techniques like state feedback and observer design.

2) Discrete representation: in control engineering refers to a mathematical description of a system or process that operates in discrete time steps rather than continuously. This representation is often used to model and analyze systems that can be observed or manipulated only at specific points in time. It involves defining the system's behavior and relationships between variables over discrete time intervals rather than in a continuous manner. Discrete representations are essential for digital control systems and computational simulations of dynamic systems. We have two types of discrete representation:

- **Difference Equation Representation:** frequently used in control engineering to model systems that progress in distinct time steps. And describes how a discrete-time system behaves by using equations that show the connections between current and past inputs, outputs, and states. This equation details how the system's variables alter between consecutive time steps. Essentially, this representation mirrors the discrete version of continuous-time systems' differential equations. It's a tool for analyzing and designing control systems in a digital setting.

$$y[k] = a \cdot y[k-1] + b \cdot u[k-1]$$

- **Discrete transfer function representation:** used to describe the relationship between the input and output of a discrete-time linear time-invariant (LTI) system. It is commonly used in control systems and signal processing to analyze and design systems that operate in discrete time steps.

The general form of a discrete transfer function is given by:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_m z^{-m}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}}$$

Where:

- $H(z)$ is the discrete transfer function.
- $Y(z)$ is the Z-transform of the output signal.
- $X(z)$ is the Z-transform of the input signal.
- $b_0, b_1, b_2, \dots, b_m$ are the feedforward (numerator) coefficients of the transfer function.
- $a_0, a_1, a_2, \dots, a_m$ are the feedback (denominator) coefficients of the transfer function.
- $z^{-1}, z^{-2}, \dots, z^{-n}$ are the discrete-time delays.

- **Discrete state-space representation:** is a mathematical model used in control engineering to describe the behavior of a dynamic system over discrete time intervals. It’s the discrete version of continuous state-space representation and consists of two main components:

- State Equations: These equations define how the state of the system evolves from one time step to the next. They express how the current state variables are influenced by the previous states, control inputs, and disturbances.

- Output Equations: These equations describe how the system's outputs are related to its current and past states. They provide a way to relate the internal state of the system to the observed outputs.

$$x[k+1] = A \cdot x[k] + B \cdot u[k]$$

$$y[k] \ = \ C \cdot x[k] \ + \ D \cdot u[k]$$

3) **Block Diagram Representation:** is a graphical method consists of blocks that represent system components, and arrows that depict the flow of signals or information between these components. Each block typically corresponds to a specific function or operation, and the interconnections between blocks represent how the components influence each other's behavior.

Block diagrams provide a visual way to understand the structure and behavior of complex control systems. They help in analyzing how signals propagate through the system, how inputs affect outputs, and how different components interact.

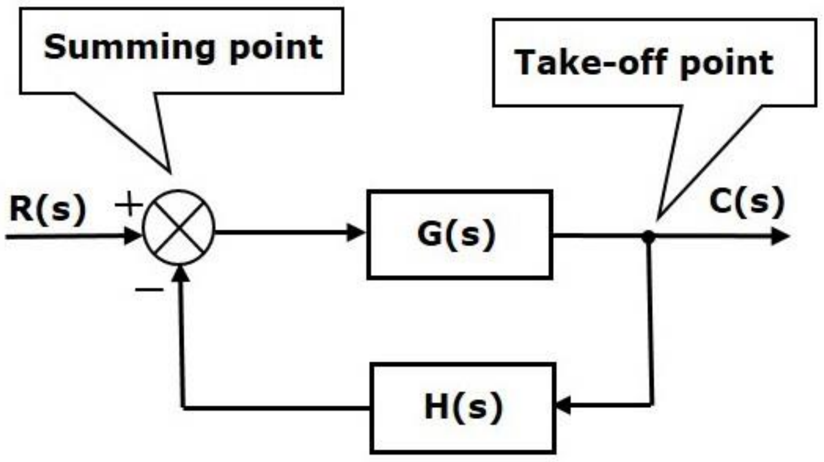
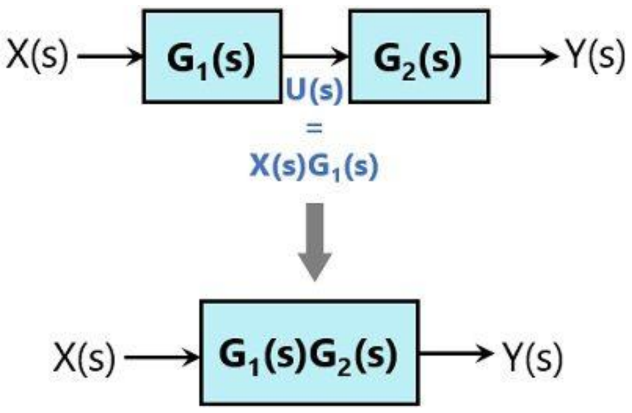


Figure 01: Block Diagram Representation of a closed loop.

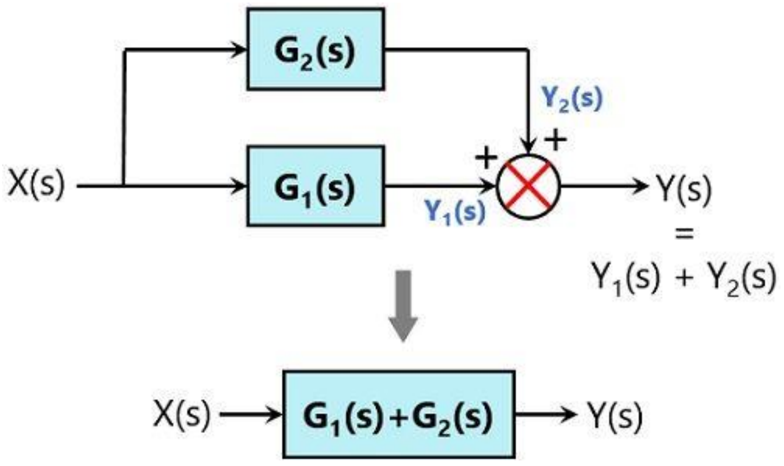
- **Block diagram rules:**

Are guidelines used to help understanding the relationships between signals and components in the system and make the analysis and design process more manageable. Here are some common block diagram rules:

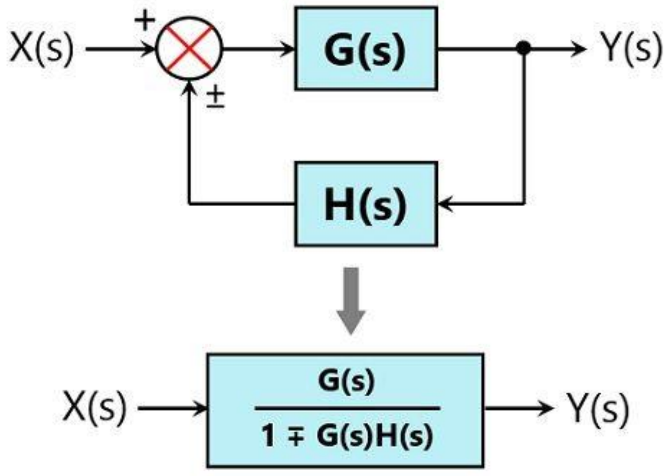
- Series Connection Rule: When two or more blocks are connected in series (one after another), their transfer functions can be multiplied to obtain the overall transfer function of the combined system



- Parallel Connection Rule: When two or more blocks are connected in parallel (input signal splits and passes through multiple paths), their transfer functions can be added to obtain the overall transfer function of the combined system.



- Feedback Connection Rule: In a feedback loop, where the output signal is fed back and combined with the input signal, the overall transfer function of the loop can be determined using specific formulas that account for the feedback.



Laplace transform:

Is a mathematical tool used to transform a function of time into a function of a frequency or complex variable.

Given a function $f(t)$ defined for $t \geq 0$, its Laplace transform $F(s)$ is defined as:

$$F(s) = L[f(t)] = \int_0^{\infty} e^{(-st)} f(t)\, dt$$

Here, s is a complex variable and represents a frequency domain variable.

Z-Transform:

It is analogous to the Laplace transform for continuous-time systems but is tailored for discrete-time signals and systems.

Given a discrete-time signal $x[n]$, where n is an integer representing the discrete time index, its Z-transform $X(z)$ is defined as:

$$X(z) = \sum_{n=-\infty}^{+\infty} x[n] \cdot z^{-n}$$

Here, z is a complex variable, and the Z-transform takes the discrete-time sequence $x[n]$ and transforms it into a function of z, which is often written in terms of z^{-1} to account for the discrete time shifts.

Representation Transformation: involve changing the representation of a system from:

1) **Differential equation to transfer function:** by Applying Laplace transform to the differential equation, or using Laplace transform table.

2) **Differential equation to difference equation:** There are three methods that involves replacing the derivatives.
   - The backward difference:
   $$\frac{dx(t)}{dt} \approx \frac{x[n] - x[n-1]}{T}$$
   - The forward difference:
   $$\frac{dx(t)}{dt} \approx \frac{x[n+1] - x[n]}{T}$$
   - Trapezoidal method:
   $$\frac{dx(t)}{dt} \approx \frac{x[n+1] - x[n-1]}{T}$$

3) **Difference equation to discrete transfer function:** by Applying z-transformation on the difference equation, or using z-transform table.

4) **Differential equation to state-space representation:** There are different methods such as Direct method and canonical forms. It depends on states that we need to study on our system.

5) **Transfer function to state-space representation:** There are different methods and it depends on the states that we need to study on our system. For the opposite, to go from the state-space representation to transfer function we use this equation:

$$G(s) = [C(sI - A)^{-1}B + D]$$

Where:

- G(s) is the transfer matrix, which is a matrix of transfer functions. It relates each input to each output.
- s is the complex frequency variable.
- A is the state matrix of the system in state-space representation.
- B is the input matrix of the system in state-space representation.
- C is the output matrix of the system in state-space representation.
- D is the feedforward matrix of the system in state-space representation.
- I is the identity matrix.

6) **Difference equation to discrete state-space representation:** can be found by manipulating the difference equation to obtain the state-space matrices. This involves expressing the current output $y[k]$ in terms of the state variables $x[k]$ and using past state variables to represent $x[k]$ in terms of $x[k-1], x[k-2]$, and so on.

7) **Transfer function to discrete Transfer function:** can be found by applying the inverted Laplace on the transfer function and then the z-transform to the obtained equation. There are second method where we can apply one of the commonly used methods such as Zero-Order Hold (ZOH) or the Tustin (Bilinear) methods.

8) **State-representation to discrete state-space representation:** can be done by applying these two equations:

$$x[k+1] = e^{AT_s}x[k] + (e^{AT_s} - I)A^{-1}Bu[k]$$

$$y[k] = Cx[k] + Du[k]$$

System Response types: there are several types of response commonly studied:

- <u>Transient Response:</u> This refers to the behavior of the system immediately after a change in the input or initial conditions. It includes features such as rise time, settling time, overshoot, and damping.
- <u>Steady-State Response:</u> This represents the long-term behavior of the system after it has settled and reached a stable condition. It's often characterized by the system's output values when the input remains constant.

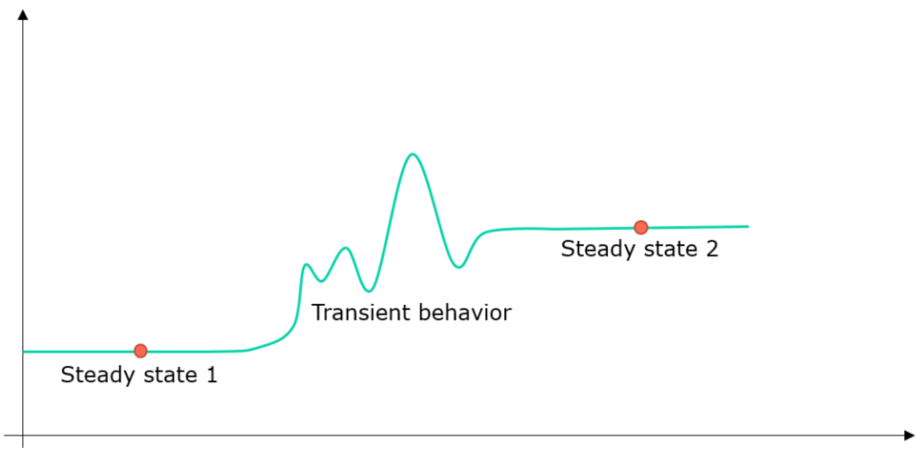


Figure 02: transient and steady-state response.

- Frequency Response: This describes how the system's output varies with different frequencies of the input signal.

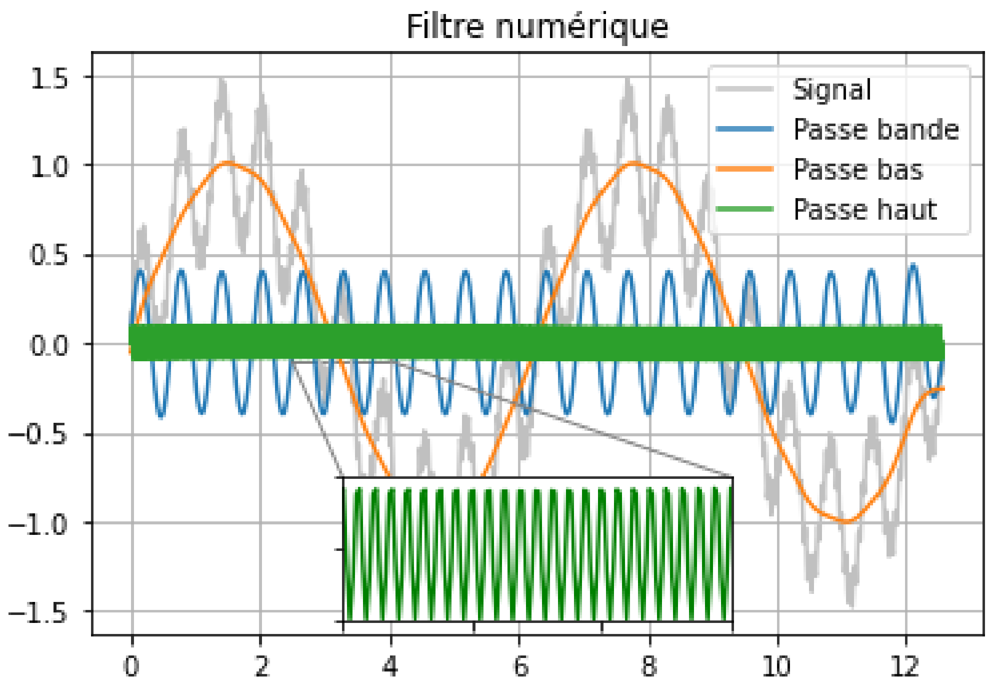


Figure 03: frequency response.

- Step Response: This is a type of transient response that describes how the system reacts to an abrupt change in input.
- Impulse Response: This is the system's response to an impulse input (a very short and intense signal). It's often used to determine the system's behavior over time.

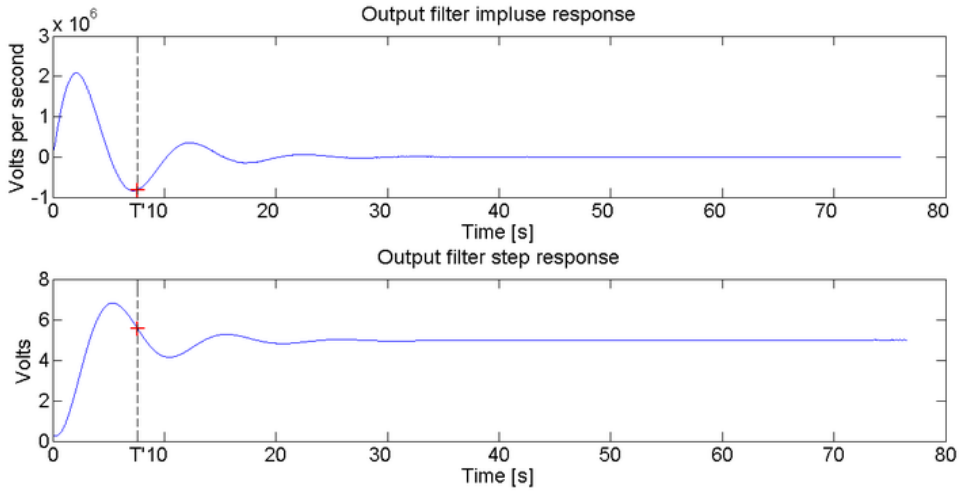


Figure 04: impulse and step response.

Note: We can derive the mathematical equations that represent the output of each of these response types as functions of time. There are various methods for solving different response types, and these methods depend on the system's representation, which will be covered later.

System response's mathematical solution:

1) **Differential equation solution:** refers to the determination of the functions or variables that describe the behavior or evolution of the system over time. A dynamic system is often described using one or more differential equations that relate the rates of change of system variables to their current values. The solution of these differential equations provides a mathematical representation of how the system's variables change as time progresses, allowing us to understand and predict the system's behavior under different conditions.
   The solutions of a differential equation in a system depend on both its order and the interplay between the forced and free parts. The order signifies the highest derivative present in the equation, influencing the complexity of the solution. The forced part represents external influences or inputs affecting the system, while the free part depicts the system's inherent behavior without external forces. Various methods, including numerical techniques and analytical approaches, are employed to solve such equations.

2) **State-space representation solution:** refers to the determination of the values of the system's state variables as they evolve over time according to a set of state-space equations. In this representation, a dynamic system is described using a set of first-order ordinary differential equations that relate the derivatives of the state variables to the state variables themselves and the system inputs. The solution of these state-space equations provides a complete description of the system's behavior by specifying the values of the state variables at any given time and how they change with time.

   To find the solution of a state-space representation using the transition matrix, you can follow these steps:

   - Formulate the State Equation: Start with the state equation in its general form:

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

     Where $\dot{x}(t)$ is the derivative of the state vector $x(t)$ with respect to time, A is the state matrix, B is the input matrix, and $u(t)$ is the control input vector.

- Calculate the Transition Matrix: $\Phi(t)$, which is the matrix exponential of the state matrix A multiplied by the time interval t:

$$\Phi(t) = e^{At}$$

Note: that the matrix exponential function can be computed using various numerical and analytical methods or software tools.

- Initial Condition: Specify the initial condition of the system, which is the state vector at an initial time t_0:

$$x(t_0) = x_0$$

- Calculate the Solution: Given an initial time t_0, the time interval t, the initial state vector x_0, and the control input vector $u(t)$, you can calculate the state vector $x(t)$ at time t using the following equation:

$$x(t) = \Phi(t - t_0)\, x_0 + \int_{t0}^{t} \Phi(t-\tau)\, B\, u(\tau)\, d\tau$$

Here, $\Phi(t - t_0)$ represents the transition matrix raised to the power of $(t - t_0)$, and the integral term accounts for the effect of control inputs over the interval (t_0, t).

- Solve the Integral: Depending on the nature of the control input u(t), you might need to solve the integral term analytically or numerically. If the control input is piecewise constant or follows a known function, you can use appropriate techniques to compute the integral.
- By following these steps, you can use the transition matrix to find the solution of the state-space representation. The transition matrix captures the system's dynamics and allows you to predict how the state of the system evolves over time in response to initial conditions and control inputs. This method is widely used in control theory, system analysis, and engineering applications to understand and predict the behavior of dynamic systems.

Stability:

The stability of a system refers to its behavior over time and its response to disturbances or changes in its initial conditions. There are several stability types that categorize how a system behaves under different conditions. Here are the main stability types:

- Stable System:

A stable system is one where the output remains bounded when subjected to bounded inputs or disturbances.

In a stable system, any small perturbations introduced to the initial conditions or input signals result in a response that gradually dies down over time.

- Unstable System:

An unstable system is one where the output grows unbounded when subjected to bounded inputs or disturbances.

In an unstable system, even small perturbations can cause the system's response to increase indefinitely, leading to undesirable behavior.

- Asymptotically Stable System:

An asymptotically stable system is both stable and, as time progresses, its response approaches a specific equilibrium point.

In other words, if the system starts from some initial conditions away from the equilibrium point, it will eventually converge to that equilibrium point as time goes to infinity.

Linear Systems Stability Analysis:

- Transfer Function Analysis: For stability analysis using transfer functions, the system is stable if and only if all the poles of the transfer function have negative real parts. This criterion ensures that the exponential terms in the inverse Laplace transform decay over time.
- Eigenvalue Analysis: Linear systems are often analyzed by examining the eigenvalues of the system's matrix representation (A in the state-space representation). If all eigenvalues have negative real parts, the system is stable.
  - ✓ Eigenvalues and Eigenvectors: For a given square matrix A, an eigenvalue (λ) is a scalar that represents how the matrix scales the corresponding eigenvector (v). In other words, when A is multiplied by v, the result is a scaled version of v. Mathematically, it is represented as $A\,v = \lambda\,v$.
  - ✓ Characteristic Equation: To find eigenvalues, you solve the characteristic equation, which is obtained by subtracting λ times the identity matrix from A and then taking its determinant. The equation is $det(A - \lambda I) = 0$, where I is the identity matrix.
  - ✓ Eigenvalue Computation: Solving the characteristic equation yields the eigenvalues of the matrix. Each eigenvalue corresponds to a set of eigenvectors that satisfy the equation $A\,v = \lambda\,v$. These eigenvectors can be calculated by solving the system of linear equations $(A - \lambda I)\,v = 0$.
  - ✓ Stability Analysis: if all eigenvalues have negative real parts, the system is stable.
- Routh-Hurwitz Criterion: This criterion is used to determine the stability of a linear system by analyzing the signs of coefficients in a characteristic polynomial.
- Bode and Nyquist Plots: These graphical tools help analyze the stability of linear systems in the frequency domain, particularly for feedback control systems.

Note: When a system's eigenvalues are zeros, we can say that the system is marginally stable (undamped).

State-space controllability:

Controllability refers to the ability to steer the system from any initial state to any desired final state in a finite amount of time using suitable control inputs. A system is considered "completely controllable" if all of its state variables can be controlled independently. Mathematically, a system is completely controllable if the controllability matrix is full rank. The controllability matrix is constructed using the system matrices A and B.

$$C = [B;\ AB;\ A^2B;\ \dots;\ A^{(n-1)}B]$$

$$det(C) \neq 0$$

If the determinant of the controllability matrix is non-zero, then the system is completely controllable. If it's zero, then the system may not be completely controllable.

State-space observability:

Observability asks whether all the internal state variables of a system can be determined or estimated from the knowledge of its output variables over a finite time interval. Mathematically, a system is considered "completely observable" if the observability matrix is full rank. The observability matrix is constructed using the system matrices A and C.

$$O = [C;\ CA;\ CA^2;\ \dots;\ CA^{(n-1)}]$$

$$det(O) \neq 0$$

If the determinant of the observability matrix is non-zero, then the system is completely observable. If it's zero, then the system may not be completely observable.

Note: We need always start by testing controllability and observability before confirming that we can control and estimate all the stat in a system.

Graphical representation:

1. **Bode:** is a graphical representation commonly used in the analysis of linear time-invariant (LTI) systems, it provides insights into the frequency response characteristics of a system. Bode plots are often divided into two separate plots: one for the magnitude and another for the phase.
2. **Nyquist:** is closely related to the Bode plot and provides valuable insights into system behavior, particularly in the context of feedback control systems. It's used to analyze the frequency response in the complex plane, showing the relationship between the frequency of an input sinusoidal signal and the magnitude and phase of the output signal. It is often used to assess the stability of a system by examining its encirclement of the critical point (-1, j0) on the complex plane.

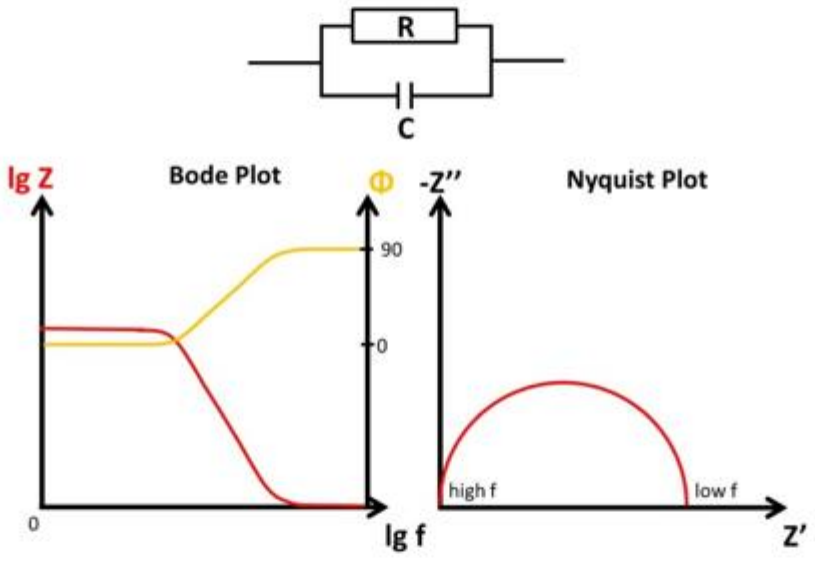


Figure 05: Bode and Nyquist diagram.

3. **Root Locus:** The root locus is a graphical representation used in control systems engineering to analyze how the locations of the closed-loop poles change as a system parameter (often a proportional gain of a controller) is varied (covered later in the control chapter).

4. **Response analysis:** involves studying how a system behaves in response to a step input, which is a sudden change from one constant value to another.

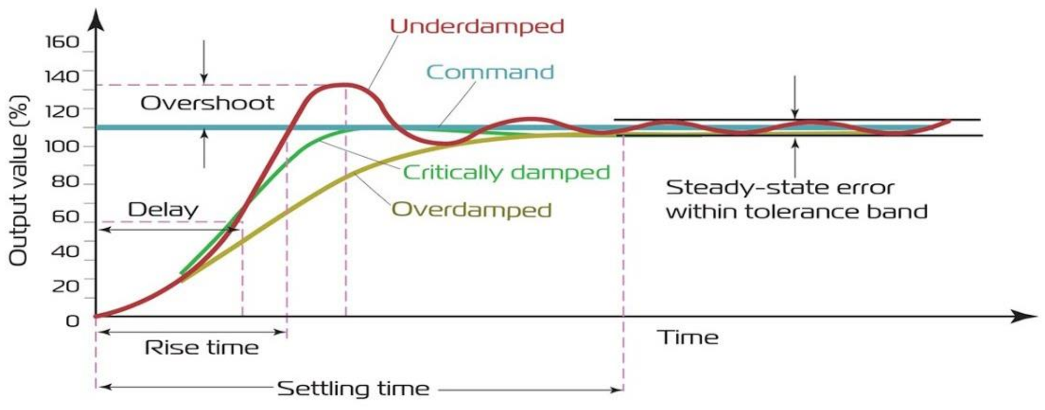


Figure 06: Response analysis.

- Rise Time: Rise time is the time it takes for the system's output to rise from 10% to 90% of its final value. It gives an indication of how quickly the system starts responding to the step input.
- Settling Time: Settling time is the time it takes for the system's output to settle within a certain tolerance of the final value. It provides insight into how long it takes for the system to stabilize after the input change.
- Overshoot: Overshoot is the percentage by which the system's output exceeds the final value before settling. It indicates the degree of oscillation or instability in the response.
- Steady-State Value: The steady-state value is the final output value that the system reaches once it has settled. It helps assess whether the system has reached its desired operating point.
- Damping: Damping refers to how quickly the oscillations in the response decay over time. Systems with high damping tend to settle more quickly without excessive oscillations.
- Natural Frequency: In cases where the system response exhibits oscillations, the natural frequency of the system can be estimated from the oscillation period.
- steady-state error: is the difference between the desired final value and the actual output when the system has reached steady state.

1. Underdamped: An underdamped system response exhibits oscillations before settling to its final value. This type of response occurs when the damping ratio ζ is less than 1. The oscillations in the response can be desirable in certain control systems, but excessive oscillations can lead to instability or poor performance.

2. Critically Damped: A critically damped system response returns to its steady-state without oscillations as quickly as possible. This occurs when the damping ratio ζ is exactly 1. Critically damped systems have a fast response without overshooting the steady-state value, which can be important for systems that need rapid settling.
3. Overdamped: An overdamped system response also returns to its steady-state without oscillations, but the rate of return is slower compared to the critically damped case. This occurs when the damping ratio ζ is greater than 1. Overdamped systems tend to have a more gradual and controlled response without the risk of oscillations.

Second order system characteristics:

To calculate the parameters of a second-degree polynomial transfer function from the overshoot D, damping ratio ζ, and pseudo natural frequency ω_{ps} (pseudo pulsation), we can use the following formulas. The transfer function can be represented as:

$$G(s) = \frac{{\omega_n}^2}{s^2 + 2\zeta\omega_n s + {\omega_n}^2}$$

Where:

- ω_n is the undamped natural frequency.
- ζ is the damping ratio.
- s is the complex frequency variable.

The pseudo natural frequency ω_{ps} is related to the actual natural frequency ω_n by:

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

The overshoot D is related to the damping ratio ζ by the following equation:

$$D = e^{-\frac{\pi\zeta}{\sqrt{1-\zeta^2}}}$$

Solving for ζ, we can get:

$$\zeta = \sqrt{\frac{\ln(D)^2}{\pi^2 + \ln(D)^2}}$$

Finally, we can calculate ω_n using the pseudo natural frequency ω_{ps} and the damping ratio ζ:

$$\omega_n = \frac{\omega_{ps}}{\sqrt{1-\zeta^2}}$$

Non-Linear Systems

Non-Linear Systems: A non-linear system is a mathematical model that describes a system whose behavior cannot be entirely represented by linear relationships between its variables. In non-linear systems, the relationships between variables are more complex and can involve products, powers, or other non-linear functions.

State Space Definition of a Non-Linear System: The state space representation is a mathematical framework used to describe and analyze the behavior of dynamic systems, including both linear and non-linear systems. In a state space representation, a system's behavior is described by a set of first-order ordinary differential equations. For a non-linear system, the state space representation takes the form:

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

$$y(t) = h(x(t), u(t))$$

Where:

- $x(t)$ represents the state vector that contains the system's state variables.
- $u(t)$ represents the input vector.
- $y(t)$ represents the output vector.
- f is a non-linear function that defines how the state variables change over time based on their current values and inputs.
- h is a non-linear function that defines how the outputs are related to the state variables and inputs.

Linearization of a Non-Linear System: Linearization is a technique used to approximate the behavior of a non-linear system around a specific operating point (often called an equilibrium point) by considering the system's linearized dynamics. The linearized representation is useful for understanding small deviations from the equilibrium and for applying linear control theory techniques.

The linearization process involves calculating the Jacobian matrix, which represents the partial derivatives of the system's state vector and output vector with respect to the state vector. The Jacobian matrix is evaluated at the equilibrium point. The resulting linearized state space representation is in the form of:

$$\Delta\dot{x}(t) = A \cdot \Delta x(t) + B \cdot \Delta u(t)$$

$$\Delta y(t) = C \cdot \Delta x(t) + D \cdot \Delta u(t)$$

Where:

$\Delta x(t)$ represents the deviation of the state variables from their equilibrium values.

$\Delta u(t)$ represents the deviation of the input variables from their equilibrium values.

A, B, C, and D are matrices derived from the Jacobian matrix.

Equilibrium Points of a Non-Linear System: An equilibrium point of a non-linear system is a state at which the system's state variables do not change with time. Mathematically, for a non-linear system:

$$f(x_{eq}, u_{eq}) = 0$$

This implies that the state variables are not changing ($\dot{x}(t) = 0$) when the inputs are held constant ($u(t) = u_{eq}$).

Phase Plane of a Non-Linear System: The phase plane is a graphical representation of the behavior of a two-dimensional non-linear system. It involves plotting the state variables against each other on a two-dimensional plane. The trajectories of the system's state variables over time create curves or patterns in the phase plane, giving insights into the system's behavior, stability, and possible solutions.

The phase plane is particularly useful for understanding the dynamics of non-linear systems, identifying equilibrium points, determining stability properties, and predicting how the system's behavior changes as initial conditions or parameters vary.

Nonlinear Systems Stability Analysis:

- Lyapunov Stability Analysis: This method involves finding a Lyapunov function that proves the stability of the system. Here are the key details of Lyapunov stability analysis for nonlinear systems:
  - ✓ **Lyapunov Functions:**

    A Lyapunov function is a scalar function $V(x)$, where x represents the state variables of the system.

    The Lyapunov function must be continuous, positive definite, and radially unbounded (i.e., it goes to infinity as x goes to infinity).

    It quantifies how far the system's state is from an equilibrium point.
  - ✓ **Lyapunov Stability Theorems:**

    Lyapunov's theorem demonstrates that the energy in the system is bounded so the system is stable.

    Lyapunov's First Theorem: If a Lyapunov function $V(x)$ exists and is such that its derivative $\dot{V}(x)$ is negative semidefinite (i.e., $\dot{V}(x) \leq 0$ for all x), then the equilibrium point is stable.

    Lyapunov's Second Theorem: If in addition to the first theorem's condition, $\dot{V}(x)$ is negative definite (i.e., $\dot{V}(x) < 0$ for all x except at the equilibrium point), then the equilibrium point is asymptotically stable.
- Linearization: Linearizing a nonlinear system around an equilibrium point can allow stability analysis using methods developed for linear systems, like eigenvalue analysis and the Routh-Hurwitz criterion. However, this method is limited to analyzing stability around small perturbations (in this case, if the system is marginally stable, we can say that the second term in the Taylor series dominates the stability and we don't know if it's stable or not. So, we apply the Lyapunov stability theorem).
- Phase Plane Analysis: This graphical method involves plotting the system's state variables against each other to visualize the system's behavior, including equilibrium points, limit cycles, and other trajectories.

Multivariable systems

Single input and single output system (SISO):

- SISO systems have a single input and a single output.
- They deal with a single input signal affecting a single output signal.

Multiple inputs and multiple outputs (MIMO):

- MIMO systems have multiple input and multiple output channels.
- They involve multiple input signals affecting multiple output signals simultaneously.

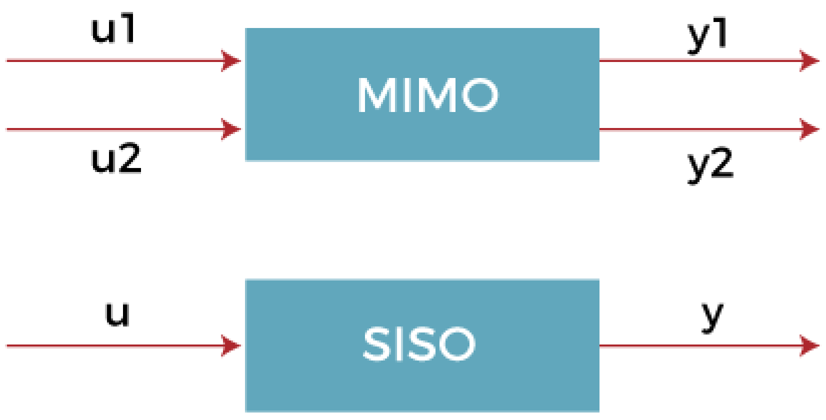


Figure 07: MIMO and SISO systems.

MIMO System representation:

1. **Transfer matrix representation:** also known as a Transfer Function Matrix or System Matrix, is a representation used to describe the relationship between inputs and outputs in a Multi-Input Multi-Output (MIMO) linear time-invariant (LTI) system. It is a matrix of transfer functions that relates the inputs to the outputs in the frequency domain.

   For a MIMO system with "m" inputs and "n" outputs, the transfer matrix representation can be expressed as follows:

   $$Y(s) \; = \; G(s) \cdot U(s)$$

   Where:

   - $Y(s)$ is an n-dimensional vector of Laplace transforms of the output signals.
   - $U(s)$ is an m-dimensional vector of Laplace transforms of the input signals.
   - $G(s)$ is an $n \times m$ matrix of transfer functions.

The elements of the transfer matrix $G(s)$ represent the transfer functions between individual input-output pairs. Each element $G_{ij}(s)$ represents the transfer function from the i-th input to the j-th output.

In general, the transfer function $G_{ij}(s)$ can be expressed as a ratio of polynomials in the Laplace variable s:

$$G_{ij}(s) = \frac{N_{ij}(s)}{D_{ij}(s)}$$

Where $N_{ij}(s)$ is the numerator polynomial and $D_{ij}(s)$ is the denominator polynomial associated with the transfer function from the i-th input to the j-th output.

2. **State-space representation:** for a MIMO system, the state-space representation can be written as follows:

State Equations:

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

Where:

$x(t)$ is an n-dimensional vector of state variables.

$\dot{x}(t)$ represents the time derivative of $x(t)$.

A is an $n \times n$ matrix containing the system's state transition dynamics.

B is an $mn \times m$ matrix relating the inputs u(t)u(t) to the state variables.

Output Equations:

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

Where:

- $y(t)$ is an pp-dimensional vector of outputs.
- C is a $p \times n$ matrix relating the state variables to the outputs.
- D is a $p \times m$ matrix that represents the direct feedthrough from inputs to outputs.
- In this representation, n represents the number of state variables, mm represents the number of inputs, and pp represents the number of outputs.

For more details about controllability and observability and analytical calculation for different representation you can check the pdf file: " cours SYSTEMES LINEAIRES MULTIVARIABLES.pdf "

This pdf contains also details about designing a controller and an observer system.

System control

System control in control engineering entails the formulation of control strategies, algorithms, and components to guarantee the desired response of a dynamic system to inputs, all while achieving precise performance objectives. In simpler terms, system control encompasses the design of controllers capable of furnishing control signals to a dynamic system. The aim is to uphold the system's response to desired inputs while achieving specific performance criteria such as accuracy, speed, stability, robustness, and more.

Designing a controller system involves a series of distinct steps, which encompass:

- System Modeling: Develop a mathematical model that accurately represents the behavior of the system to be controlled. This can involve differential equations, transfer functions, or state-space representations, depending on the complexity of the system.
- Control Objectives: Define the control objectives, such as stability, tracking performance, disturbance rejection, and robustness. These objectives guide the design process.
- Controller Selection: Choose an appropriate control strategy based on the system's characteristics and control objectives. Common strategies include PID control, state feedback control, and advanced techniques like adaptive or optimal control.
- Controller Design: Design the control law or algorithm that computes the control inputs based on the system's current state or error. This involves tuning parameters, designing compensators, and ensuring stability through techniques like root locus, Bode plots, and Nyquist analysis.
- Simulation and Analysis: Simulate the designed control system using software tools to analyze its performance, stability, and response to various scenarios. This step helps in fine-tuning the controller and identifying potential issues.
- Implementation: Implement the control algorithm in hardware or software. This could involve programming microcontrollers, PLCs (Programmable Logic Controllers), or software platforms.
- Testing and Validation: Test the control system in real-world conditions to ensure it behaves as intended. Adjust controller parameters if necessary to achieve desired performance.
- Tuning: Refine the control parameters to optimize the system's performance, considering factors like overshoot, settling time, and steady-state error.
- Robustness Analysis: Evaluate the control system's robustness against uncertainties, disturbances, and variations in system parameters. Apply techniques like sensitivity analysis or robust control design to enhance stability under different conditions.
- Iterative Process: Control system design is often iterative, involving adjustments, simulations, and testing cycles to achieve the desired performance and robustness.
- Documentation: Document the design process, controller parameters, simulation results, and test outcomes for future reference and potential improvements.

Overall, control system design aims to create a control strategy that effectively achieves the desired objectives while maintaining stability, performance, and adaptability in the face of changing conditions. In this chapter we will cover different control strategies from conventional ones to advanced and Ai-based strategies.

Open Loop control system:

An open-loop control system, also known as a non-feedback control system, operates without using feedback to adjust its output. In this type of system, the control action is determined solely based on the input reference and the system's dynamics. The controller generates a control signal that directly influences the process or plant being controlled, without considering the actual output or its effect on the system.

Characteristics of open-loop systems:

- Lack of corrective action based on system output.
- Susceptible to disturbances and variations in the process.
- Limited accuracy and reliability, especially in the presence of uncertainties.
- Typically used in applications where precision is not critical, or where the system dynamics are well understood and constant.

Closed loop control system (Feedback control):

A closed-loop control system, also known as a feedback control system, incorporates feedback from the system's output to adjust the control action. This type of system compares the actual output with the desired reference input and uses the difference (error) to adjust the control signal, aiming to minimize the error and bring the system output closer to the desired value.

Characteristics of closed-loop systems:

- Incorporates feedback for error correction.
- More robust against disturbances and uncertainties.
- Able to maintain system stability and desired performance.
- Suitable for applications requiring accuracy and precise control.

Components of a closed-loop control system:

- Sensor/Transducer: Measures the system's actual output or relevant variables.
- Controller: Computes the control action based on the error between the reference and the actual output.
- Actuator: Executes the control action by adjusting the system's inputs.
- Process/Plant: The system being controlled.
- Feedback Path: The path through which the actual output is fed back to the controller.

Traditional control strategies:

A. Proportional-integral-derivative (PID) control:

PID control, also known as Proportional-Integral-Derivative control, is a widely used feedback control algorithm in engineering and control systems. It's a type of closed-loop control that aims to regulate a system's output by adjusting the control input based on the error between the desired setpoint and the actual measured process variable.

The PID controller uses three components to calculate the control output:

- **Proportional (P) Term**: The proportional term generates a control output that is proportional to the current error. It aims to reduce the error by applying a control action that is directly related to the magnitude of the error. A higher error leads to a stronger control action. The proportional gain (Kp) determines the sensitivity of the control system to the error.

- **Integral (I) Term**: The integral term integrates the accumulated error over time and generates a control output to eliminate any steady-state error. It addresses issues such as offset and bias in the control system. The integral term helps in handling situations where the proportional control alone cannot eliminate the residual error over time. The integral gain (Ki) determines the rate at which accumulated error affects the control action.
- **Derivative (D) Term**: The derivative term anticipates the future behavior of the error by considering its rate of change. It helps in dampening the control action to prevent overshoot and oscillations in the system's response. The derivative term provides a control action that is proportional to the rate of change of the error. The derivative gain (Kd) determines the contribution of the rate of change of the error to the control action.

The PID controller's control output is calculated as follows:

$$Control\ Output\ =\ Kp\ *\ (Error)\ +\ Ki\ *\ \int (Error)\ dt\ +\ Kd\ *\ d(Error)/dt$$

Where:

$Error$: The difference between the desired setpoint and the actual measured process variable.

Kp: Proportional gain.

Ki: Integral gain.

Kd: Derivative gain.

$\int (Error)\ dt$: Integral of the error over time.

$d(Error)/dt$: Rate of change of the error.

The three components (P, I, and D) work together to provide a well-balanced control response. The proper tuning of the PID gains is crucial to achieve desired control performance, stability, and responsiveness. Different processes may require different tuning methods to optimize the PID gains for the best performance.

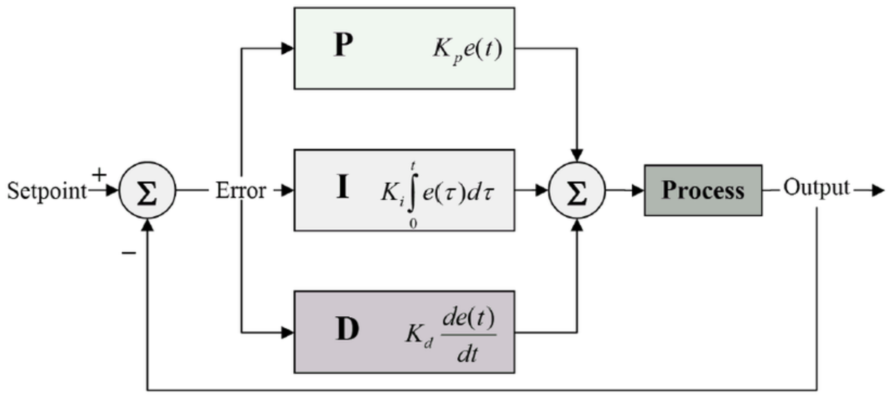


Figure 08: PID control block diagram.

B. Pole placement control:

Pole placement control is a technique used in control engineering to design a control system by strategically placing the poles of the closed-loop system's transfer function. This method is employed to achieve specific performance characteristics and stability properties in the controlled system.

In pole placement control, the goal is to place the poles of the closed-loop system's transfer function in desired locations in the complex plane. By doing so, the control engineer can shape the system's response according to specific requirements, such as settling time, overshoot, and stability. This technique is particularly useful when designing control systems with specific performance objectives.

We have MATLAB RLTOOL which is a graphical interface for designing a pole placement controller and testing different poles positions in the complex plane for a simulated dynamic system.

C. Partial and total compensation control:

Compensation control is a technique used in controller design to compensate for the effects of the numerator and denominator of a system's transfer function. In the case of total compensation control, the controller is often designed as the inverse of the system's transfer function, with an additional delay to account for causality constraints. This approach can be applied when the poles and zeros of the system are located in regions with negative real parts, ensuring stability.

Conversely, in partial compensation control, only specific parts of the system are compensated, usually focusing on stabilizing the numerator.

D. Internal model control (IMC):

Internal Model Control (IMC) is a control strategy used to design controllers that incorporate an internal model of the desired system behavior. The primary objective of IMC is to make the controlled system's response closely resemble the behavior of this internal model. IMC is particularly useful when you want precise tracking of reference signals, disturbance rejection, and robust performance. The key ideas of an IMC:

- Reference Model: The reference model defines the desired behavior of the controlled system. It specifies how the system should respond to changes in setpoints and disturbances. The reference model serves as a target for the controller to mimic.
- Control Structure: IMC involves designing both the controller and the process in a way that the controlled system's behavior matches the reference model. The controller is designed to enforce the reference model's behavior on the system.
- Servo and Regulatory Action: IMC distinguishes between servo and regulatory actions. Servo action ensures that the controlled system responds correctly during transients, while regulatory action maintains steady-state accuracy.

For more understanding. Let's consider this example, a first-order system with transfer function:

$$G(s) = \frac{1}{\tau s + 1}$$

Where: τ is the time constant.

1. Step 1: Define the Desired Internal Model

The first step is to define the desired internal model. Let's assume that we want the closed-loop system to behave like a first-order system with a desired time constant τ_d and gain of 1. The desired transfer function of the internal model is:

$$M(s) = \frac{1}{\tau_d s + 1}$$

2. Step 2: Calculate Controller Transfer Function

The controller transfer function $C(s)$, can be calculated as the ratio of the desired internal model to the process transfer function:

$$C(s) = \frac{M(s)}{G(s)} = \frac{\frac{1}{\tau_d s+1}}{\frac{1}{\tau s+1}} = \frac{\tau}{\tau_d} \frac{1}{\frac{\tau}{\tau_d}s+1}$$

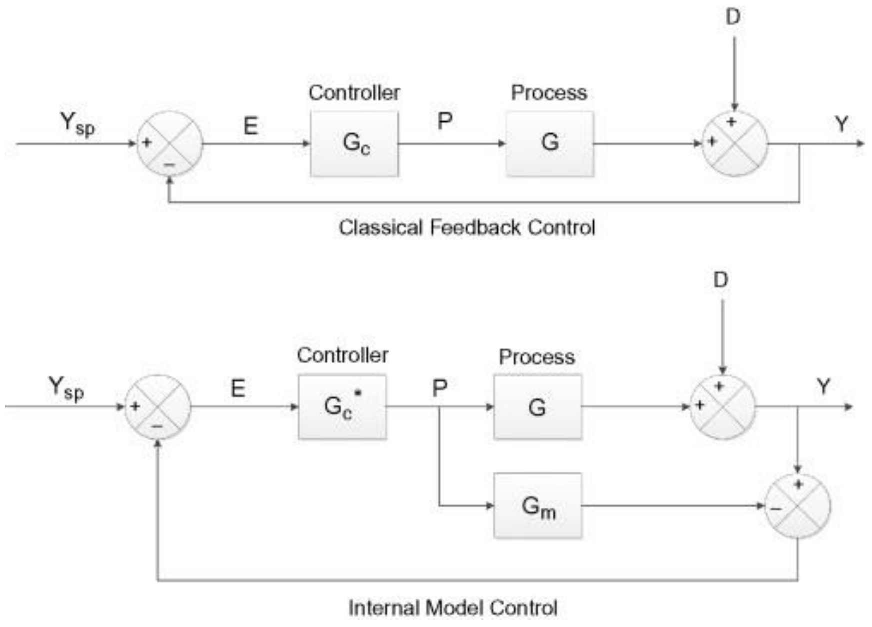


Figure 10: Internal model and classical feedback control.

Remarque: we need to take in consideration the stability of the system’s numerator. In the case of instable zeros, we need just to take only stable part of the numerator.

Advantages of IMC:

- Precise Tracking: IMC is particularly effective at achieving accurate tracking of reference signals and setpoints.
- Disturbance Rejection: The internal model helps reject disturbances by ensuring that the controlled system responds as if it's following the internal model's behavior.
- Customization: IMC allows for tailoring the controller design to specific performance requirements and control objectives.

E. Reference-Servo-Tracking “RST” control:

RST control is a control strategy used in the design of digital controllers for discrete-time systems. This controller represents the three key stages in the design process. RST control is particularly useful for achieving accurate tracking of reference signals while maintaining stability and robustness.

The reference model defines the desired closed-loop response of the system. It specifies how the system should behave in terms of transient response, settling time, overshoot, and other performance criteria.

The servo design phase focuses on designing the controller to ensure that the system closely follows the reference model during transient conditions. It involves adjusting controller parameters to achieve a desired transient response.

The tracking design phase ensures that the controlled system accurately tracks changes in the reference signal over time. The goal is to minimize tracking error and maintain steady-state accuracy.

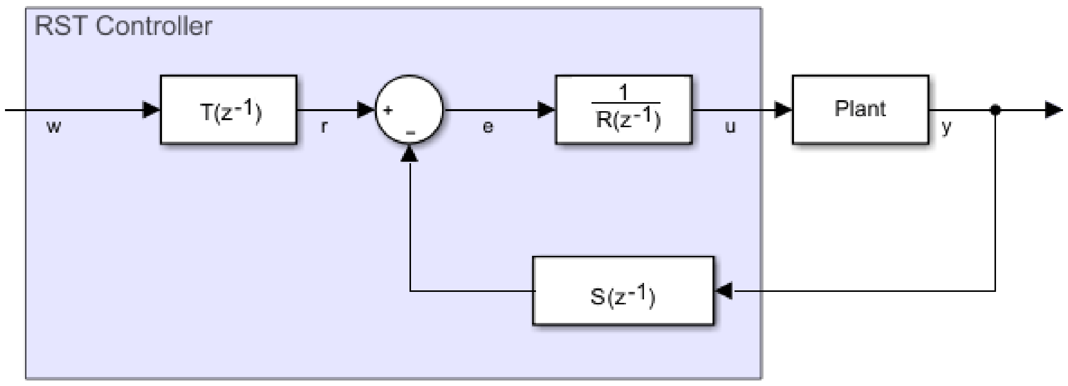


Figure 11: RST control.

By expressing the closed loop equation and make it equal the desired system and also by applying the Diophantine equation we can find the S, R and T polynomial parameters.

F. Smith Predictor:

The Smith Predictor is a control strategy used to improve the control of systems with significant time delays. It's particularly useful for processes with long dead times that can negatively impact the stability and performance of traditional control approaches. The Smith Predictor essentially compensates for the effects of time delays in the system, allowing for more effective control.

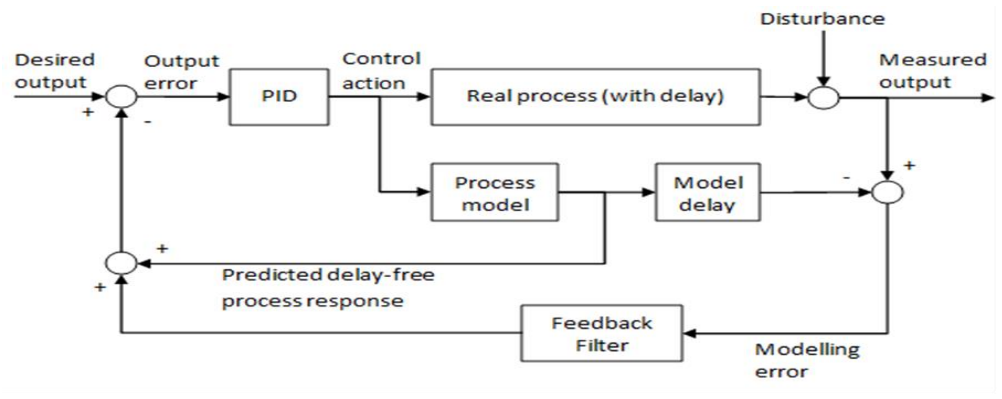


Figure 12: Smith predictor block diagram.

Some of the key Concepts and Components of the smith predictor:

- Time Delay: In many real-world systems, there can be delays between the application of a control action and the corresponding effect on the process output. These delays can lead to instability or poor control performance.
- Predictive Controller: The Smith Predictor introduces a predictive controller that estimates the future behavior of the system's output based on the current and past control actions. This estimation is used to compensate for the time delay.

The basic idea of the Smith Predictor is to use an internal model of the process dynamics to predict the future output of the system. This prediction is then used to generate control signals that counteract the effects of the time delay. Here's how it works:

- Internal Model: An internal model of the process is created based on the known process dynamics. This model mimics how the process responds to control actions.
- Prediction: The predictive controller calculates the expected future output based on the current and past control actions using the internal model.
- Compensation: The control signal is adjusted using the predicted output, effectively compensating for the time delay.

Advantages of the Smith Predictor:

- Time Delay Compensation: The primary advantage is its ability to compensate for time delays, improving stability and performance.
- Reduced Oscillations: By predicting and compensating for future behavior, the Smith Predictor can reduce oscillations and improve transient response.
- Improved Control Performance: Processes with long dead times can be difficult to control using conventional methods. The Smith Predictor can significantly enhance control performance in such cases.

G. Adaptative Control:

Adaptive control is a sophisticated control methodology employed in dynamic systems where uncertainties, parameter variations, and external disturbances are present. Unlike traditional control methods that assume fixed system parameters, adaptive control continuously adjusts the controller's parameters in response to changing conditions. The core idea is to ensure that the control system can maintain desired performance levels and stability, even when the system's characteristics are subject to fluctuations. This is achieved through real-time parameter estimation, adaptation algorithms, and feedback mechanisms that allow the controller to modify its behavior to match the evolving system dynamics.

1. Model Reference Adaptative Control (MRAC):

Model Reference Adaptive Control (MRAC) is a specific type of adaptive control strategy that centers around the concept of a reference model. In MRAC, a reference model is designed to represent the desired behavior of the system. The control algorithm continually compares the output of the actual system with the output of the reference model. By adjusting its parameters based on the error between these two outputs, the controller aims to make the controlled system emulate the behavior of the reference model. MRAC is especially useful in scenarios where precise knowledge of system dynamics is lacking, as it focuses on guiding the system's behavior towards a predefined standard, even in the face of uncertainties and changes.

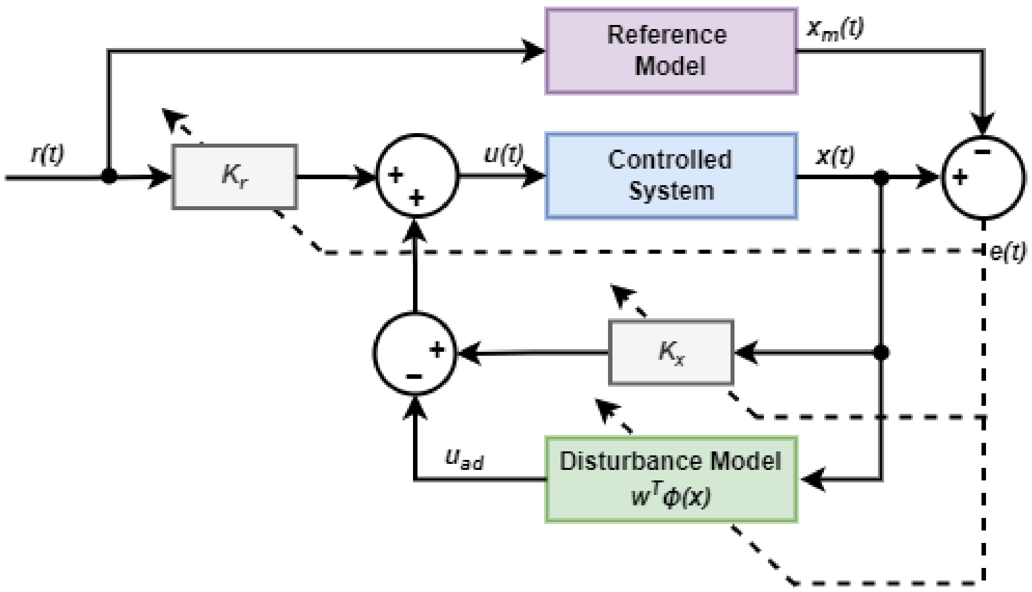


Figure 13: MRAC block diagram.

2. Self-Tuning Regulator (STR):

The Self-Tuning Regulator (STR) is an adaptive control technique that emphasizes the real-time estimation of the actual system's parameters based on observed input-output data. The controller adjusts its own parameters using these parameter estimates, aiming to maintain optimal or near-optimal performance. By adapting both the controller's parameters and the estimated system parameters, STR offers a way to manage uncertainties and variations while striving to achieve effective control in dynamic environments. This makes STR particularly well-suited for systems with complex and changing dynamics where precise model information might be limited.

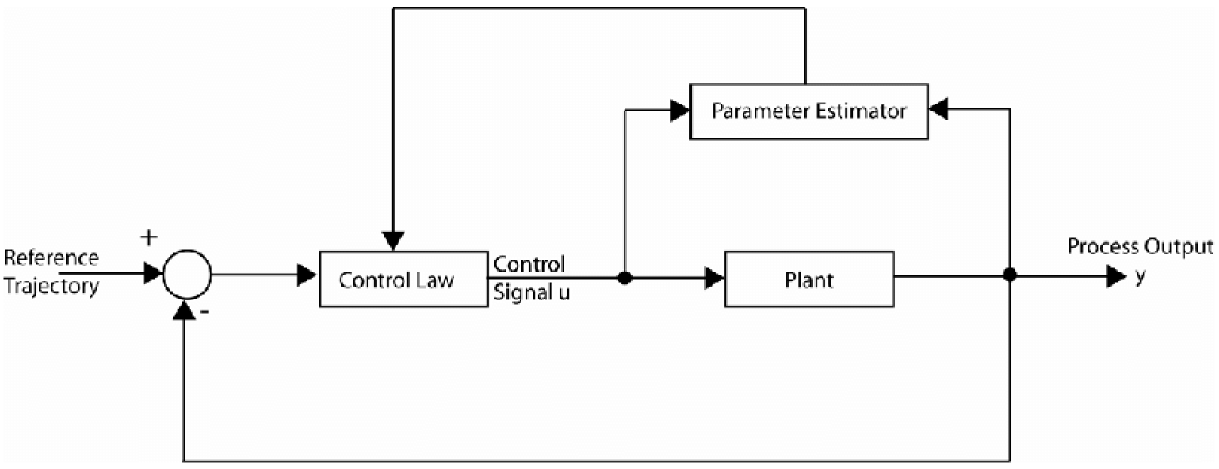


Figure 14: STR control block diagram.

Remarque: There are different adaptive control methods, and algorithms. You can check details of some of these methods such as the MRAC-MIT and MRAC-HYPERSTABILITY in the following pdf file: “chap_1_comm_Adapta.pdf”.

H. Predictive Control -Optimal control-:

Predictive control, also known as Model Predictive Control (MPC), is an advanced control strategy used in various fields of engineering and industrial processes. It is a control approach that makes use of a predictive model of the system being controlled to optimize control inputs over a finite future time horizon. The basic idea is to solve an optimization problem at each time step to find the optimal control inputs that will minimize a certain cost function while satisfying system constraints.

Here are the key details about predictive control:

- Control Objective and Cost Function: The main goal of predictive control is to optimize the behavior of a dynamic system over a finite prediction horizon by selecting control inputs. This is done by formulating a cost function that represents the desired performance of the system. The cost function typically includes terms that penalize deviations from desired setpoints, control effort, and other performance criteria.
- Prediction Model: Predictive control relies on a predictive model of the system. This model describes the dynamic behavior of the system, usually in the form of differential equations or difference equations. The model can be based on first principles, data-driven approaches, or a combination of both.
- Time Horizon: Predictive control operates over a finite time horizon that consists of a sequence of future time steps. The length of the time horizon determines how far into the future the control algorithm considers when making decisions. Longer time horizons provide better prediction but can also lead to more complex and computationally intensive calculations.
- Control Horizon: Within the time horizon, a shorter "control horizon" is often used. This refers to the number of future time steps for which control inputs are calculated explicitly. At each time step, the control inputs are applied only for the first step of the control horizon, and the optimization problem is solved again at the next time step with updated information.
- Constraints: Predictive control can handle various types of constraints such as input saturation, output limitations, state constraints, and more. These constraints are incorporated into the optimization problem to ensure that the calculated control inputs do not violate any limits.
- Online Optimization: At each time step, predictive control involves solving an optimization problem to find the optimal control inputs over the prediction horizon while considering the current state of the system. This can be computationally demanding, especially for complex systems with long prediction horizons. Advanced optimization algorithms are often used to efficiently solve these problems.
- Adaptability: Predictive control can adapt to changes in the system dynamics or constraints by repeatedly solving the optimization problem at each time step. This allows the controller to adjust its behavior in real time based on new measurements and changing conditions.
- Advantages: Predictive control offers advantages such as the ability to handle complex dynamics, constraints, and disturbances, as well as the capability to optimize multiple objectives simultaneously.

For more details and different predictive models and its application, check these two PDF files:

“chap_2_Commade_prédictive.pdf”.

“chapitre2_commande prédictive5_AAV.pdf ”.

State-space control strategies

A. State feedback control (Controller & Observer):

State feedback control is a technique used in control systems to regulate the behavior of a dynamic system by manipulating its state variables. It involves the use of both a controller and an observer.

- **Controller (State Feedback Controller)**: The controller computes a control input based on the current state of the system. The goal of the controller is to calculate control signals that can manipulate the system's state variables to achieve desired performance or stability. In state feedback control, the controller typically employs a feedback loop that uses the current state information to adjust the control inputs. The design of the controller is often based on the system's state-space representation, which describes the relationship between the system's state variables, control inputs, and outputs.
- **Observer (State Estimator or Kalman Filter)**: An observer, also known as a state estimator or Kalman filter, is used to estimate the unmeasured state variables of the system based on the available measurements. Since not all state variables may be directly measurable, the observer helps provide an estimate of the complete state vector using the available sensor measurements and the system's dynamics. This estimated state information is then used by the state feedback controller to calculate the control inputs. The observer's design involves estimating the state variables while minimizing the effects of noise and disturbances.

The combined operation of the state feedback controller and the observer is referred to as the state feedback control system. This approach is particularly useful for controlling systems with multiple states and when precise control of the system's behavior is required.

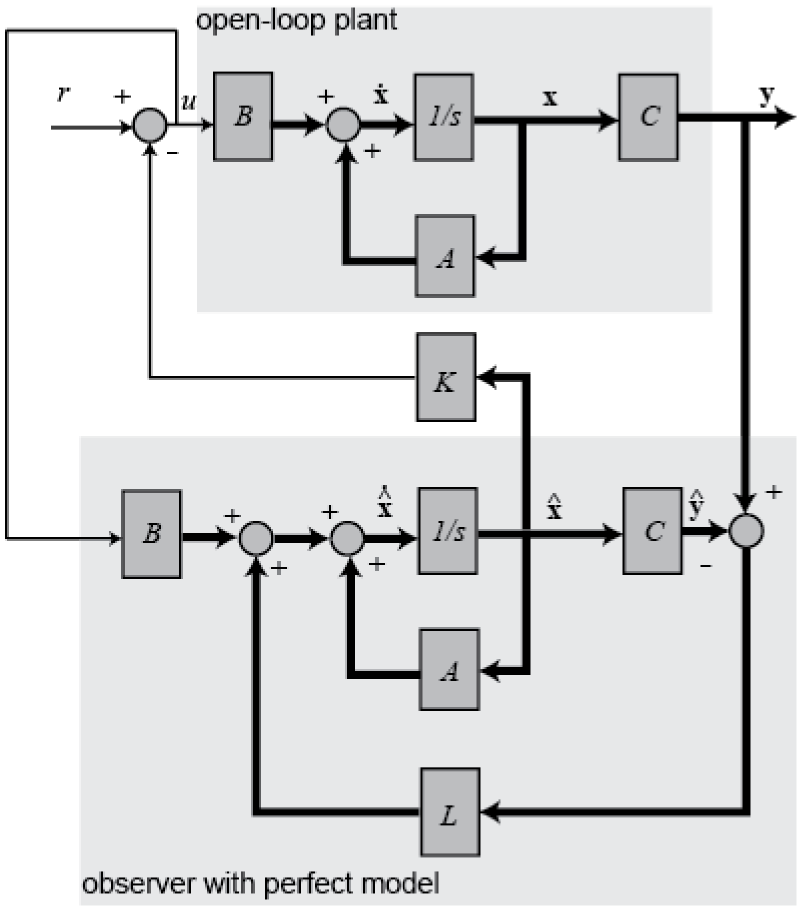


Figure 09: State-space control block diagram.

Check this PDF file for analytical calculations details: " cours SYSTEMES LINEAIRES MULTIVARIABLES.pdf "

B. LQR & LQE & LQG -Optimal control-:

Optimal control is a branch of control theory that focuses on finding control strategies that minimize a specific performance criterion while taking into account system dynamics and constraints. The goal is to design a control law that guides a system to achieve desired behavior while optimizing some objective, such as minimizing energy consumption, maximizing efficiency, or reducing errors.

Three common approaches to optimal control are the Linear Quadratic Regulator (LQR), Linear Quadratic Estimator (LQE) and the Linear Quadratic Gaussian (LQG) control methods.

Linear Quadratic Regulator (LQR):

The Linear Quadratic Regulator (LQR) is a fundamental optimal control technique primarily applied to linear time-invariant systems. The goal of LQR is to design a control law that minimizes a quadratic cost function while ensuring system stability.

Components of the LQR problem:

- State-Space Representation: The system is represented in state-space form, where the dynamics are described by a set of linear differential or difference equations.
- Cost Function: The cost function to be minimized is typically defined as a quadratic function of the system's state and control inputs. It represents a trade-off between regulating the state variables and controlling the inputs.
- Weighting Matrices: The LQR problem involves defining two positive semidefinite matrices, the state (error) weighting matrix (Q) and the control weighting matrix (R). These matrices control the relative importance of the state (error) and control effort terms in the cost function.
- Control Law: The optimal control law is computed by solving the algebraic Riccati differential equation, which gives the optimal state feedback matrix (K) that minimizes the cost function. The control law is given by $u = -Kx$, where u is the control input and x are the state vector.

Note: that the difference between the state feedback and the LQR control, is that stat feedback finds the feedback matrix (K) based on the desired dynamic. On the other hand, LQR finds the (K) based on the optimization (minimization) of an objective function.

In MATLAB we have two commands:

- For state feedback we have "place(A,B,eigns).
- For LQR we have "LQR(A,B,Q,R).

LQR stat-space equation:

$$\dot{x} = (A - k_r B) \cdot x$$

This equation can be solved with defining a desired system or an objective function.

Linear Quadratic Estimator -Kalman Filter- (LQE):

Linear Quadratic Estimation (LQE), often referred to as the Kalman Filter, is a control theory technique used to estimate the state of a dynamic system based on a set of noisy measurements. Just like Linear Quadratic Regulation (LQR) deals with controlling a system by minimizing a quadratic cost function, LQE addresses the problem of state estimation by minimizing the expected quadratic error between the estimated state and the true state, considering both the system dynamics and measurement noise.

While LQR focuses on designing control inputs to regulate a system's behavior, LQE concentrates on designing an optimal estimation process to determine the system's current state, even when measurements are noisy. Both LQR and LQE involve the concept of optimization and rely on the principles of linear systems theory. LQR optimizes control inputs, while LQE optimizes state estimates.

LQE stat-space equation:

$$\dot{e} = \left(A - k_f C\right) \cdot e$$

This equation can be solved with defining a desired system or an objective function.

Linear Quadratic Gaussian (LQG):

LQG control combines the concepts of LQR and LQE to design optimal controllers for systems with both state feedback and state estimation components. LQG control aims to minimize both the control effort and the estimation error. It employs the LQR control law for the control input and the Kalman filter for state estimation. This approach provides a comprehensive control strategy that takes into account both control and estimation uncertainties.

LQG control optimally balances control effort and state estimation accuracy to achieve better system performance and stability.

In conclusion, LQR, LQE, and LQG are fundamental concepts in optimal control theory, addressing control, estimation, and combined control-estimation problems for linear dynamic systems. These techniques are widely used in various fields, including engineering, robotics, economics, and more.

Check this YouTube playlist to understand analytical calculation of LQR, LQE and LQG:

https://www.youtube.com/watch?v=s_9InuQAx-g&list=PLMrJAkhIeNNR20Mz-VpzgfQs5zrYi085m&index=18

C. Sliding Mode Control (SMC) -Applied to non-linear systems-:

One of the most crucial issues in automated industrial plants is the control of dynamic systems, particularly robots and complex systems, with non-linearities, modeling uncertainties, and disturbances like dynamic parameters and effects. Consequently, adopting robust control approaches, such as sliding mode control, has advanced significantly.

Sliding mode control is a particular type of variable structure system. The basic idea behind this control strategy is to bring the trajectory of the system's state variable, regardless of the initial conditions, onto a hyper-surface known as a sliding surface which represents the desired dynamical behavior in the phase plane (reaching mode) and to maintain it on the surface until it tends towards the origin of that plane (the sliding mode).

The sliding mode control law synthesis involves three main steps which are:

Sliding surface design

The general form of the sliding surface that was proposed by Slotine is given by:

$$S(x) = \left(\frac{\partial}{\partial x} + \lambda\right)^{n-1} e(x)$$

where:

- n represents the system's order.
- x is the state's vector.
- $e(x)$ is the error between the current variable x and the desired one x_d.
- λ is a positive coefficient representing the slope of the sliding surface.

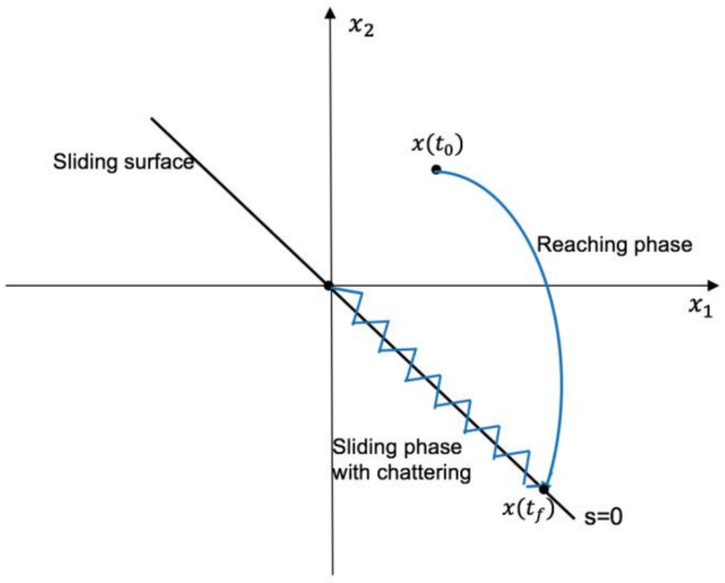


Figure 15: Trajectory Modes in The Sliding Surface.

In our case, $n = 2$. Therefore, the surface's equation becomes:

$$S(x) = \dot{e} + \lambda e$$

Where:

$$e = q - q_d = x_1 - x_{1d}$$

And:

$$\dot{e} = \dot{q} - \dot{q_d} = x_2 - x_{2d}$$

The convergence and existence conditions:

The conditions that are commonly considered in sliding mode analysis are:

- A Lyapunov candidate function must be chosen in order to guarantee the convergence and the stability of the state's trajectory on the sliding surface. We usually opt for the following function:

$$V(x) = \frac{1}{2}S^2$$

Whose first derivative is:

$$\dot{V}(x) = \dot{S}(x) \cdot S(x)$$

- To guarantee the existence and convergence of the sliding mode on the surface, the following equation must be verified:

$$\dot{S}(x) \cdot S(x) < 0$$

Determination of the control law:

The sliding mode control employs a switching control law to direct the system's state trajectory onto the sliding surface. It consists of two components:

$$u = u_{eq} + u_n$$

- Continuous control law: The equivalent control term, denoted as u_{eq}, aims to eliminate the non-linearities and uncertainties present in the system dynamics and is in charge of directing the state trajectory of the system along the sliding surface. It is issued from:

$$\dot{S} = 0$$

- Discontinuous control law: when the state of the system deviates from the sliding surface, the discontinuous control law is triggered. In order to make sure the system follows the required reference trajectory; it generates a control action that brings back the system's state to the sliding surface. It is derived from the condition:

$$\dot{S} = -K \cdot sgn(S)$$

where:

- K is a positive definite diagonal matrix.
- sgn is a function defined by:

$$sng(S) = \begin{cases} +1 \; if \; S(x) > 0 \\ -1 \; if \; S(x) < 0 \end{cases}$$

This PDF contains more details about this control law and its application: "chap 7 mode glissant.pdf".

The chattering phenomenon:

Sliding mode control essentially employs discontinuous control laws to drive the state trajectory of the system on a sliding surface and confine it within its vicinity. However, it exhibits high-frequency oscillations known as Chattering when the system state reaches the sliding surface as shown in the figure 15. This causes negative effects on the controlled system and activates unwanted dynamics.

The system can achieve smoother control action, lessen components' wear and tear, and enhance the overall system performance and stability by reducing or eliminating chattering. To mitigate it, various techniques can be used:

- Replacing the sign function that forms the discontinuous component of the control law by a function that can be thought of as an approximation to the former one such as the sigmoid function or hyperbolic tangent function.
- Using a high-order Sliding Mode approach.
- Using the Boundary Layer approach introduced by Slotine and Li.

D. Backstepping control -Applied to non-linear systems-:

Backstepping control is a nonlinear control technique used to stabilize and control complex systems with nonlinear dynamics. It's particularly useful for systems where the traditional linear control techniques are inadequate due to strong nonlinearities, uncertainties, or changing operating conditions. Backstepping control breaks down the control problem into a series of steps, with each step stabilizing a subset of the system's dynamics.

Here's an overview of the backstepping control technique:

1. Basic Concept: Backstepping control is designed to transform a complex nonlinear system into a set of simpler subsystems, each of which can be stabilized individually. The key idea is to introduce virtual control inputs that stabilize the system dynamics step by step, "backstepping" through the system.
2. Steps in Backstepping: The backstepping technique involves the following main steps:
3. Choice of Lyapunov Function: For each subsystem introduced, a Lyapunov function is selected. The Lyapunov function helps in analyzing the stability properties of the subsystem.
4. Design of Virtual Control Laws: Virtual control inputs are designed for each subsystem to ensure that the Lyapunov function decreases along the system trajectories. These virtual control inputs are chosen such that they steer the system's states toward the desired equilibrium points.
5. Stabilization of Subsystems: Starting from the last subsystem, the control inputs are designed recursively, considering the virtual control inputs from the previous steps. This process continues until the original system's dynamics are stabilized.

Extensions and Variations: There are variations and extensions of backstepping control:

- Adaptive Backstepping: Incorporates adaptation to handle uncertainties in the system's parameters.
- Integral Backstepping: Addresses the integration of integral terms in the control law to improve tracking performance.
- Output Feedback Backstepping: Allows the use of output measurements instead of full state feedback.

For more details about this method and its application check this PDF file: "chap8_commande par Backstepping_AAV.pdf".

AI-Based control strategies:

A. Fuzzy Logic control:

Fuzzy Logic Control (FLC) is an AI based control methodology that deals with handling uncertainty and imprecision in system behavior. It's particularly useful in situations where traditional binary control methods are not well-suited due to the complexity of the system or the ambiguity of the input data.

Fuzzy Logic Control is based on the principles of fuzzy set theory. In contrast to classical binary logic, where variables are either true or false (1 or 0), fuzzy logic allows for degrees of truth between 0 and 1. This enables FLC to model and reason with vague or ambiguous information, making it suitable for systems with imprecise inputs and outputs.

The main components of a Fuzzy Logic Control system include:

- Fuzzification: This step involves converting crisp input data into fuzzy linguistic terms. For example, instead of using "high," "medium," and "low" as plain numerical values, these terms are defined as fuzzy sets that can describe the membership of an input value in each of these categories.
- Fuzzy Rules: These are a set of conditional statements that define how the input variables relate to the output variables. Each rule consists of antecedents (input conditions) and a consequent (output action). These rules are usually expressed in terms of linguistic variables and their corresponding fuzzy sets.
- Inference Engine: The inference engine evaluates the fuzzy rules based on the fuzzy values of the input variables. It combines these rules to generate fuzzy output sets.
- Fuzzy Aggregation: This step combines the output fuzzy sets generated by the inference engine to obtain a single aggregated fuzzy set that represents the overall output.
- Defuzzification: In this final step, the aggregated fuzzy set is transformed back into a crisp value that represents the control action to be taken. Various defuzzification methods can be used, such as center of gravity, maximum membership, or mean of maximum.

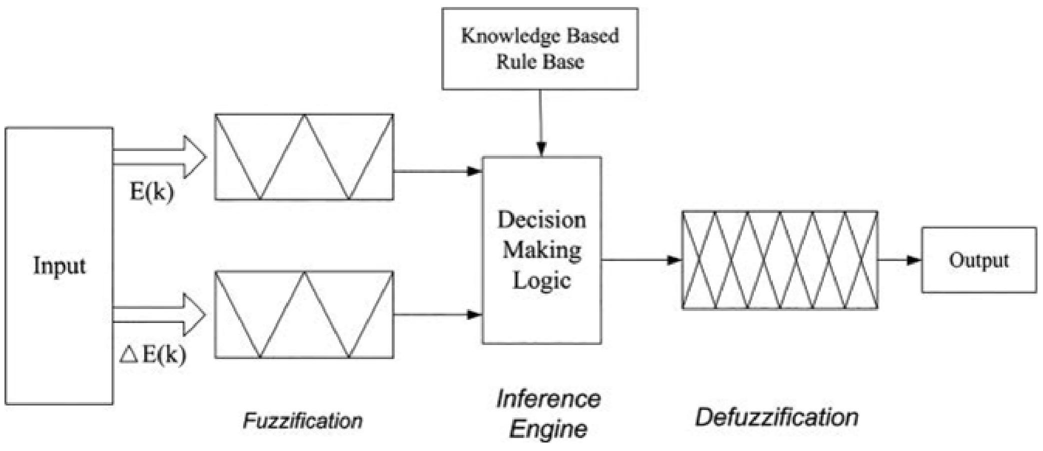


Figure 16: Fuzzy Logic Control.

Check this PDF for more details about FLC and its application:
“chap_5_commande_logique_floue.pdf”

B. Neural Network Control:

Neural networks are powerful tools for controlling dynamic systems due to their ability to learn complex relationships from data and adapt to changing conditions. They have been widely used in various fields for controlling dynamic systems such as robotics, process control, autonomous vehicles, and more. Here's a general overview of how neural networks can be used for controlling dynamic systems:

- Modeling Dynamic Systems: Neural networks can be used to model the behavior of dynamic systems by learning the underlying dynamics from input-output data. This involves training the network to predict future states or outputs based on past inputs and system states. Depending on the complexity of the system, different types of neural architectures can be used, such as recurrent neural networks (RNNs), long short-term memory networks (LSTMs), or more advanced variants like transformers (Check neural network by Iheb BOUARICHE.pdf).
- Controller Design: Neural networks can be used to design controllers that map system states to control inputs. For example, Proportional-Integral-Derivative (PID) controllers, for instance, can be replaced or augmented with neural network controllers for improved adaptability.
- Adaptive Control: Dynamic systems often face changing operating conditions or uncertainties. Neural networks are well-suited for adaptive control, where they can continuously learn and update their internal representations to accommodate these changes. This adaptability can help the controller maintain optimal performance under varying conditions.
- Model Predictive Control (MPC): MPC is a control strategy that uses a model of the system to predict future states and then optimize control inputs to achieve desired objectives. Neural network models can be integrated into MPC frameworks to improve the accuracy of predictions and enhance control performance.
- Hybrid Systems: Neural networks can be combined with traditional control techniques to form hybrid control systems. This allows leveraging the strengths of both approaches to handle different aspects of control, such as stability guarantees from traditional controllers and adaptability from neural networks.

Check this pdf file for more details about neural network application on control engineering:

" chap_6_commande_réseau_neurone.pdf "

C. Reinforcement Learning Control -Optimal control-:

Reinforcement Learning (RL): Reinforcement Learning is a machine learning paradigm where an agent learns how to interact with an environment to maximize a cumulative reward signal. The agent takes actions in the environment, receives feedback in the form of rewards, and learns to update its policy (strategy) over time to improve its decision-making process.

The RL framework consists of several key components:

- Agent: The learner or decision-maker that interacts with the environment.
- Environment: The external system with which the agent interacts and from which it receives feedback.
- State: A representation of the current situation or configuration of the environment.
- Action: Choices made by the agent that influence the environment.
- Reward: A scalar signal received by the agent after each action, indicating the immediate desirability of that action.

- Policy: The strategy or behavior of the agent that maps states to actions.
- Reinforcement Learning for Optimal Control: Reinforcement learning and optimal control share similarities, and RL can be seen as a broader framework that encompasses optimal control. In fact, many RL algorithms aim to solve optimal control problems by learning optimal policies through interaction with the environment.

RL algorithms can learn to approximate the optimal control policy. These algorithms iteratively improve the agent's policy based on the rewards received from the environment. Model-free RL methods, such as Q-learning, policy gradients, and actor-critic algorithms, can be applied.

RL's efficacy can diminish when confronted with complex tasks or systems due to challenges in handling large state and action spaces, intricate dynamics, and difficulties in exploration. This is where deep reinforcement learning (DRL) comes into play.

D. Deep Reinforcement Learning Control -Optimal control-:

Deep Reinforcement Learning (DRL) is a subfield of machine learning that combines deep learning techniques with reinforcement learning principles to enable agents to learn how to make decisions and take actions in order to maximize a reward signal. In the context of control engineering, DRL has shown promise in addressing complex control problems where traditional methods might struggle, especially when dealing with high-dimensional state and action spaces, nonlinearity, and uncertainty.

Here's how DRL is applied to control engineering:

- Agent and Environment: In DRL, there is an agent that interacts with an environment. The environment is the system that the agent seeks to control. The agent takes actions in the environment based on its observations of the current state and aims to maximize a cumulative reward signal over time.
- State and Action Spaces: The state space represents the current state of the environment, which the agent perceives through observations. The action space consists of the possible actions the agent can take to influence the environment.
- Reward Signal: At each time step, the agent receives a reward signal from the environment based on the action it took and the resulting state transition. The goal of the agent is to learn a policy—a mapping from states to actions—that maximizes the expected cumulative reward over time.
- Deep Neural Networks: DRL often employs deep neural networks, specifically deep Q-networks (DQNs) or policy networks, to approximate the agent's policy or value function. These networks enable the agent to generalize its learning across different states and actions.
- Exploration and Exploitation: DRL agents need to balance exploration (trying new actions to learn about their effects) and exploitation (choosing actions that the agent believes will lead to higher rewards). This balance is crucial for the agent to discover optimal policies.
- Training Process: During training, the agent interacts with the environment, collects experience (state, action, reward, next state), and uses this experience to update its neural network. Various algorithms like Deep Q-Networks (DQN), Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO), and Soft Actor-Critic (SAC) can be used to update the agent's policy or value function.

Deep reinforcement learning is currently a vital topic in the field of Artificial Intelligence. In addition to control engineering, it finds applications in various domains such as robotics, finance, healthcare, gaming, and autonomous systems.

Check these PDF files for all the details about RL and Deep reinforcement learning theory and its application:

"Deep Reinforcement Learning by Iheb BOUARICHE.pdf"

"Deep Reinforcement Learning Control of a Manipulator Robot (Compared to traditional control strategies) - INGENIORAT' degree-.pdf"

"Vision Based Control of a Robotic Arm -MASTER degree-.pdf"

Optimal control

Optimal control refers to a mathematical framework used to determine the best possible way to control a dynamic system in order to achieve a specific goal. This could involve finding the control inputs that minimize or maximize certain performance criteria while taking into account system dynamics, constraints, and other factors.

Convex cost function:

A convex cost function is one where any line segment connecting two points on the function's graph lies above the graph itself. Mathematically, a cost function f is convex if, for any two points x_1 and x_2 in the domain of f and for any t in the interval $[0, 1]$, the following inequality holds:

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$

In simpler terms, a convex cost function doesn't have any local minima that are higher than the global minimum. Convex optimization problems have desirable properties. They usually have a unique global minimum that can be efficiently found using a variety of optimization algorithms.

Non-Convex cost function:

A non-convex cost function is one where there exists at least one line segment connecting two points on the function's graph that lies below the graph itself. In other words, non-convex cost functions can have multiple local minima, including some that are not the global minimum. This makes optimization more challenging because conventional optimization algorithms might get stuck in local minima and fail to find the best solution.

Optimizing non-convex functions requires more sophisticated techniques. Some optimization algorithms are designed to handle non-convex problems, including reinforcement learning, genetic algorithms, and more. However, these methods might not always guarantee finding the global minimum and could be sensitive to initialization and parameters.

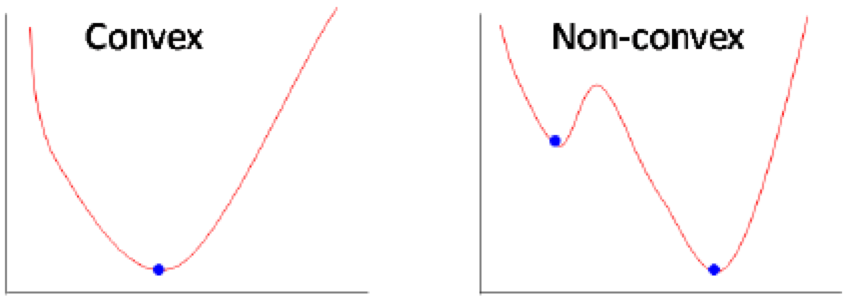


Figure 17: Convex vs non-Convex function.

Optimal control parameters:

Gradient Descent:

The most efficient technique for using Gradient Descent to optimize control problems is by optimizing predefined fixed controller parameters, such as those used in Model Predictive Control (MPC), Adaptive Control, Linear Quadratic Regulator (LQR) parameters, and others. This approach involves iteratively adjusting these parameters using the gradient information of a cost function to enhance the control system's performance.

Gradient descent is an iterative optimization algorithm that aims to find the minimum of a function by iteratively moving in the direction of steepest descent (negative gradient). Here's how we can use gradient descent to optimize the cost function of a control system:

- Step 1: Define your cost function $J(theta)$ that you want to minimize, where theta represents the control parameters.
- Step 2: Initialize the control parameters theta randomly or with some initial values.
- Step 3: Choose a learning rate alpha, which determines the step size in each iteration. It's important to choose a suitable learning rate to ensure convergence.
- Step 4: Repeat the following steps until convergence:
  A. Compute the gradient of the cost function with respect to the control parameters:

  $$gradient = \nabla J(theta)$$

  B. Update the control parameters using the gradient descent update rule:

  $$theta = theta - alpha * gradient$$

- Step 5: Monitor the convergence of the algorithm by tracking the change in the cost function over iterations. Stop when the cost function change becomes small enough or after a fixed number of iterations.

Newton-Raphson and Quasi-Newton:

Newton's method is an optimization algorithm that uses the second derivative (Hessian matrix) of the cost function in addition to the gradient information. It often converges faster than gradient descent but requires computing and inverting the Hessian matrix, which can be computationally expensive. Here's how you can use Newton's method for optimization:

- Step 1: Define the cost function J(theta) and its gradient ∇J(theta).
- Step 2: Initialize the control parameters theta.
- Step 3: Choose a stopping criterion, like a maximum number of iterations or a small change in the cost function.
- Step 4: Repeat the following steps until the stopping criterion is met:
  a. Compute the gradient of the cost function:

  $$gradient = \nabla J(theta)$$

  b. Compute the Hessian matrix of the cost function with respect to the control parameters:

  $$Hessian = \nabla^2 J(theta)$$

  c. Update the control parameters using the Newton's update rule:

$$theta = theta - (Hessian)^{-1} * gradient$$

- Step 5: Monitor convergence and stop when the chosen criterion is met.

Notes:

In practice, the Hessian matrix might be ill-conditioned or difficult to compute. In such cases, you might use approximations or regularized versions of Newton's method such as Quasi-Newton and Trust region methods.

Both gradient descent and Newton's method have their strengths and weaknesses. Gradient descent is generally simpler to implement and computationally less intensive, but it might converge slower, especially when the cost function has complex and irregular landscapes. Newton's method, on the other hand, can converge faster but might face challenges with large-scale optimization problems due to the computational cost of computing and inverting the Hessian matrix. Both methods work well in convex cost function and suffer with non-convex ones.

In both cases, proper tuning of hyperparameters, such as learning rate for gradient descent and the stopping criterion for both methods, is essential to achieve good convergence behavior. It's also important to initialize the control parameters carefully.

For solving LQR optimal control problem, it is possible to solve it using optimization methods like Newton's method or gradient descent. However, the Riccati equation provides the optimal control policy directly. And this analytical solution is more efficient and accurate than iterative optimization methods like Newton's or gradient descent when dealing with LQR problems.

Riccati equation:

Riccati equation is a key component in solving the LQR problem for linear systems. It provides a way to compute the optimal state feedback gain matrix that minimizes the cost function.

The continuous-time algebraic Riccati equation for the LQR problem is given by:

$$A^T P + P A - P B R^{-1} B^T P + Q = 0$$

Where:

- A is the system dynamics matrix.
- B is the control input matrix.
- Q is the state cost matrix (positive semi-definite).
- R is the control input cost matrix (positive definite).
- P is the symmetric positive definite matrix to be solved for. It represents the optimal state feedback gain matrix.

By solving the Riccati equation and finding P, we can then find to the optimal state feedback gain matrix K by using the following egality:

$$K = R^{-1} B^T P$$

Which is then used to calculate the control input:

$$u = -Kx$$

Note: that complex and high order systems can be solved with numerical solvers instead of analytic solving for simple systems.

Check the following YouTube video to understand the demonstration of Riccati equation:

https://www.youtube.com/watch?v=ZktL3YjTbB4

Genetic Algorithm -very recommended for non-convex cost function-:

Genetic algorithm (GA) is a powerful optimization and search technique inspired by the principles of natural selection and evolution. It is used to find approximate solutions to complex optimization and search problems. The basic idea behind a genetic algorithm is to evolve a population of potential solutions over generations to gradually improve the quality of the solutions.

Here's a detailed breakdown of how a genetic algorithm works:

- Initialization: Create an initial population of potential solutions (individuals) to the problem. Each individual is represented as a set of parameters (genes) that define a possible solution (with a binary representation).
- Evaluation: Each individual in the population is evaluated based on a fitness function. The fitness function quantifies how well an individual solves the problem. The better the individual's solution, the higher its fitness value.
- Selection: Select individuals from the current population to serve as parents for the next generation. The probability of selection is usually proportional to an individual's fitness, individuals with higher fitness values are more likely to be selected. Popular selection techniques include:
  - Roulette Wheel Selection: Individuals are selected with probabilities proportional to their fitness values.
  - Tournament Selection: Randomly select a few individuals and choose the best one from the group.
  - Rank-based Selection: Individuals are sorted by fitness, and the probability of selection is based on their rank.
- Crossover (Recombination): Create new individuals (offspring) by combining the genetic information of selected parents. This is often achieved through crossover or recombination of genes. Crossover involves selecting a crossover point in the gene sequence of parents and exchanging genetic material to create new offspring.
- Mutation: Introduce small random changes (mutations) to the genetic information of offspring. Mutation helps introduce diversity in the population, preventing the algorithm from getting stuck in local optima.
- Replacement: Replace the current population with the new offspring population. This step ensures that the population size remains constant.
- Termination (stopping criteria): Check termination conditions. These could be a maximum number of generations, a satisfactory solution, or a certain level of fitness reached.
- Repeat: If termination conditions are not met, repeat steps 3 to 7 for a specified number of generations or until a satisfactory solution is found.

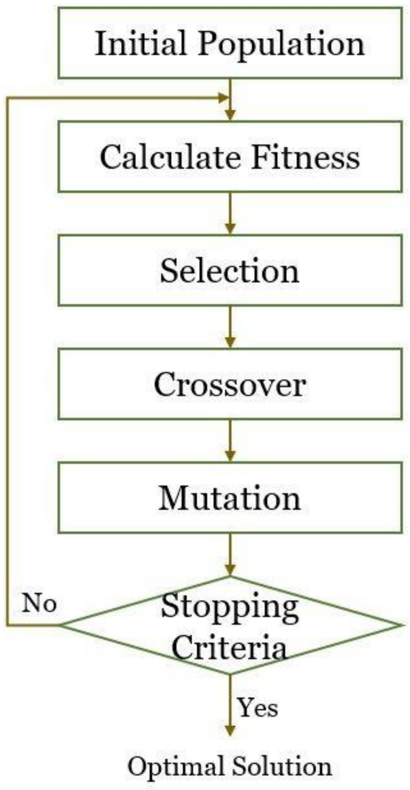


Figure 18: Genetic algorithm.

Notes:

It's important to note that genetic algorithms are heuristic methods, meaning they don't guarantee finding the optimal solution but are effective at finding good solutions in complex search spaces where other optimization methods might struggle.

One of the limitations of genetic algorithms (GAs) is that they can be computationally expensive, especially when dealing with large populations and complex fitness functions. The primary reason for this expense is the need to evaluate each individual in the population using the fitness function, which can be a time-consuming process depending on the problem being solved.

In control engineering, GA is one of the powerful techniques for optimizing controller parameters like LQR, PID and others and it's also very powerful for non-convex cost function.

This YouTube video contain an example of the GA:

https://www.youtube.com/watch?v=uQj5UNhCPuo&t=188s

Optimal control:

Dynamic programing:

Dynamic Programming is a powerful technique used in the field of optimal control, which deals with finding the best control strategy for a system to achieve a specific goal while minimizing or maximizing a certain objective function. This technique is particularly useful when the problem can be broken down into smaller subproblems that exhibit overlapping substructure.

Here's how dynamic programming is applied to optimal control problems:

- Formulate the Problem: Clearly define the problem in terms of a system's state, control inputs, dynamics, constraints, and the objective function to be optimized. This often involves differential equations that describe how the system evolves over time.
- Discretize the Problem: Continuous-time optimal control problems are often transformed into discrete-time problems by dividing time into small intervals (time steps). This results in a sequence of states and control inputs.
- Bellman's Principle of Optimality: The key idea behind dynamic programming is the principle of optimality, which states that an optimal policy for a problem with a certain initial state depends only on the optimal policies for the subproblems associated with the reachable states from that initial state. This allows the problem to be broken down into smaller subproblems.
- Define the Value Function: The value function represents the optimal cost-to-go from a given state to the goal while following the optimal control policy. It is usually denoted as $V(state)$ and depends on the current state and possibly time.
- Recurrence Relation: The value function satisfies a recursive relation known as the Bellman equation. This equation expresses the value at a current state in terms of the values at reachable successor states and the immediate cost or reward associated with the current state and control input.

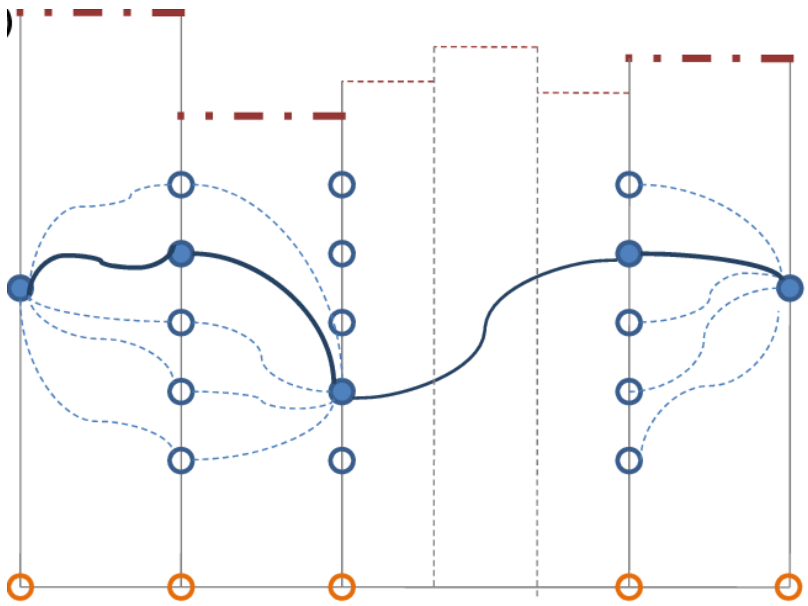


Figure 19: Dynamic programing control optimization of a discretized dynamic system.

<u>Bellman's Principle of Optimality:</u>

Consider a discrete-time dynamic programming problem with a finite time horizon. Let's denote the state at time step t as x_1 and the control (action) taken at time step t as u_t. The goal is to find a sequence of controls $u_1, u_2, \dots, u_T$ that optimizes an objective function J over the time horizon T.

The Principle of Optimality can be expressed mathematically using the Bellman equation.

The Bellman equation for this problem can be written as follows:

$$V_t(x_t) = min_{u_t}[c_t(x_t, u_t) + V_{t+1}(f_t(x_t, u_t))]$$

Where:

- $V_t(x_t)$ is the value function at time t and state x_t, representing the optimal cumulative cost from time t to the end of the horizon.
- $c_t(x_t, u_t)$ is the immediate cost (negative reward) associated with taking action u_t in state x_t at time t.
- $f_t(x_t, u_t)$ is the state transition function that describes how the system evolves from x_t to x_{t+1} given action u_t.
- $V_{t+1}(x_{t+1})$ is the value function at the next time step $t + 1$ and state x_{t+1}.

Notes:

- Dynamic programming can become computationally expensive for problems with large state or control spaces. In such cases, approximations or alternative methods like reinforcement learning might be more suitable.
- The higher order of a system dynamic and the lower discretization time step increase the complexity of the problem and becomes computationally expensive.
- This Bellman equation is the inverse of RL Bellman equation, because this minimizes the value function and in RL, we search to maximize it.
- Out of the context of control engineering, there are two methods of DP: Memorzation and Tabulation.

This YouTube video contain an example of DP applied on a dynamic system:

https://www.youtube.com/watch?v=7qRCZ9_wfYI

- DP can solve the Bellman Equation of Optimality by iteratively test all the states and actions in the environment until the value function converge.

Reinforcement and Deep Reinforcement learning -Applied to non-linear systems-:

-More Recommended than DP-

Reinforcement learning and Deep Reinforcement learning algorithms, which are covered in the System Control chapter, offer an alternative and advanced approach to Bellman's Principle of Optimality (The same Bellman equation but with maximizing an objective function and with trial-and-error technique). By utilizing discrete action and discrete state algorithms such as Temporal Difference, Q-learning, SARSA, Monte Carlo, and others, it becomes possible to address optimization challenges within discretized dynamic systems with large state and control spaces.

For an in-depth understanding of Reinforcement and Deep Reinforcement Learning theory and their practical applications, I recommend referring to the following PDF files:

"Deep Reinforcement Learning by Iheb BOUARICHE.pdf"

"Deep Reinforcement Learning Control of a Manipulator Robot (Compared to traditional control strategies) - INGENIORAT' degree-.pdf"

"Vision Based Control of a Robotic Arm -MASTER degree-.pdf"

Note: that RL and DRL are potent methodologies capable of training agents (controllers) to tackle diverse dynamic problems. These include both continuous and discrete systems, linear and non-linear dynamics, as well as scenarios with stochastic or deterministic behavior, whether the underlying models are known or unknown. The applicability of these methods is contingent on the specific algorithm employed and the computational resources and training time available.

Difference between “GD & Newton & LQR” vs Dynamic programing & RL & DRL:

First methods are optimization methods of the controller’s parameters, while “DP & RL & DRL” provide the optimal control for the system’s inputs.

Reinforcement learning can be a solution for Dynamic programming discrete problems (The Bellman Equation of Optimality) and other different problems.

Pontryagin's Maximum Principle:

Pontryagin's Maximum Principle is a fundamental concept in the field of optimal control theory, a branch of mathematics that deals with finding the best possible control strategy for a dynamical system to achieve a certain objective.

At its core, the principle provides a set of necessary conditions that must be satisfied by an optimal control and an optimal trajectory for a given dynamical system. These conditions help us determine whether a particular control strategy is optimal or not.

Here's a detailed explanation of the key components of Pontryagin's Maximum Principle:

- State Dynamics: Consider a dynamical system described by a set of ordinary differential equations (ODEs) of the form:

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

where $x(t)$ represents the state of the system at time t, $u(t)$ represents the control input at time t, and f is a function that describes the evolution of the state over time.

- Cost Function: The goal in an optimal control problem is often to minimize or maximize a cost functional, denoted by J, which depends on the state trajectory and the control inputs over a certain time interval $[t_0, t_f]$:

$$J = \phi\left(x\left(t_f\right)\right) + \int_{t_0}^{t_f} L(x(t), u(t), t)dt$$

Here, ϕ is the final cost and L is the instantaneous cost rate. The specific form of ϕ and L depends on the problem at hand. Most of the time L represent the error and control over the time.

- Hamiltonian Function: The Hamiltonian function, denoted by H, is introduced to connect the state dynamics, the cost function, and the control inputs:

$$H(x, u, \lambda, t) = L(x, u, t) + \lambda^T \cdot f(x, u, t)$$

where λ is a vector of Lagrange multipliers, known as the costate variables. The Hamiltonian function essentially combines the instantaneous cost rate with the influence of the state dynamics through the costate variables.

- Maximum Principle: Pontryagin's Maximum Principle states that if a control trajectory $u^*(t)$ is optimal, then there exist costate variables $\lambda^*(t)$ such that the following conditions are satisfied:
  a. Dynamic system condition:

$$\dot{x}(t) = \frac{\partial H}{\partial \lambda}(x^*(t), u^*(t), \lambda^*(t), t)$$

  b. Maximum Condition: At each time t and for each state variable i, the control $u^*(t)$ maximizes the Hamiltonian H over all possible controls:

$$\frac{\partial H}{\partial u_i}(x^*(t), u^*(t), \lambda^*(t), t) = 0$$

c. Costate Dynamics: The costate variables $\lambda^*(t)$ evolve according to the negation of the gradient of the Hamiltonian with respect to the state variables:

$$\dot{\lambda}^*(t) = -\frac{\partial H}{\partial x}(x^*(t), u^*(t), \lambda^*(t), t)$$

d. Transversality Conditions: At the final time t_f, the costate variables are related to the gradient of the final cost ϕ with respect to the state variables:

$$\lambda^*(t_f) = \nabla_x \phi(x^*(t_f))$$

These conditions collectively provide a way to characterize the optimal control and state trajectories for a given optimal control problem. By solving the differential equations associated with these conditions, one can determine the optimal control strategy and the corresponding state evolution that minimize (or maximize) the specified cost function.

Pontryagin's Maximum Principle is a powerful tool for solving a wide range of optimal control problems in various fields, including engineering, economics, and biology. It helps us gain insights into the structure of optimal solutions and aids in the design of effective control strategies for complex systems.

The Hamiltonian:

The Hamiltonian is an important concept because it combines the objective of reaching a certain state and satisfying constraints. The optimal control problem involves finding control inputs $u(t)$ that minimize the Hamiltonian while still obeying the state dynamics and constraints.

By introducing the co-states (multipliers) and the Hamiltonian, we create a framework that considers both the objective function and the constraints in an optimal control problem. This allows us to find a balance between achieving the desired state and satisfying the constraints, leading to a comprehensive solution.

In summary, the Hamiltonian is a central concept in optimal control theory that combines the objective function, system dynamics, and constraints. The introduction of time-varying Lagrange multipliers (co-states) helps us account for the influence of constraints on the overall optimization problem.

Note: that the Hamiltonian and the Hamilton-Jacobi equation are not the same thing.

Hamilton-Jacobi equation:

The Hamilton-Jacobi equation is a fundamental concept in optimal control theory and is used to solve problems related to finding optimal trajectories for a dynamical system while minimizing a certain cost.

Let's break down the Hamilton-Jacobi equation for optimal control in a detailed explanation:

1. Optimal Control Problem: Consider a dynamic system described by a state vector xx evolving over time t according to the equation:

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

Where $\dot{x}$ represents the time derivative of x, u is the control input applied to the system, and $f(x, u)$ is the vector field governing the system's dynamics.

The goal is to find an optimal control input $u^*(t)$ that minimizes a given cost functional J over a specified time horizon:

$$J = \phi\left(x(t_f)\right) + \int_{t_0}^{t_f} L(x(t), u(t), t)dt$$

Where $L(x, u, t)$ is the instantaneous cost (also called the Lagrangian), $\Phi(x(tf))$ is the terminal cost, and t_0 and t_f define the initial and final times.

2. Hamiltonian Function: To derive the Hamilton-Jacobi equation, we introduce the Hamiltonian function H associated with the system:

$$H(x, u, p, t) = L(x, u, t) + p^T f(x, u)$$

Where p is the costate vector, also known as the adjoint variable. The costate vector evolves over time according to the adjoint equation:

$$\dot{p} = -\frac{\partial H}{\partial x}$$

With the terminal condition:

$$p(t_f) = \frac{\partial \phi}{\partial x}(x(t_f)).$$

3. Hamilton-Jacobi Equation: The Hamilton-Jacobi equation is a partial differential equation that connects the Hamiltonian function and the optimal control problem. It takes the form:

$$\frac{\partial V}{\partial t} + H(x, u, \frac{\partial V}{\partial t}, t) = 0$$

Where $V(x, t)$ is the value function, which represents the minimum achievable cost-to-go starting from state x at time t. The equation above is a first-order nonlinear partial differential equation in V, x, u, and t.

4. Solution of the Hamilton-Jacobi Equation: Solving the Hamilton-Jacobi equation yields the value function $V(x, t)$. Once V is obtained, the optimal control input $u^*(x, t)$ can be determined by solving for u that minimizes the Hamiltonian H:

$$u^*(x, t) = argmin_u H(x, u, \frac{\partial V}{\partial t}, t)$$

5. Computing Optimal Trajectories: With the optimal control input $u^*(x, t)$ in hand, the optimal trajectory $x^*(t)$ can be computed by integrating the system dynamics:

$$\dot{x}^*(t) = f\left(x^*(t), u^*(x^*(t), t)\right)$$

With an initial condition: $x^*(t_0) = x_0$.

System identification

System identification is the process of building mathematical models that describe the behavior of a dynamic system based on observed input-output data. The goal of system identification is to understand the underlying dynamics, relationships, and parameters of a system without having explicit knowledge of its internal workings. This is especially useful in cases where it's difficult to derive a precise model from first principles, or when the system is too complex to model accurately manually.

(We have different representation of system such as ARMA and others)

- Least Squares Method (LSM): Explanation: The least squares method is a fundamental technique in system identification. It aims to find the model parameters that minimize the sum of the squared differences between the predicted and actual output values based on the given input data.
- Recursive Least Squares Method (RLSM): Explanation: The recursive least squares method is an adaptive technique that continuously updates the model parameters as new data arrives. This allows for real-time adjustment of the model to changing conditions.
- Weighted Least Squares Method (WLSM): Explanation: The weighted least squares method assigns different weights to different data points, giving more importance to certain observations. This is useful when there is heteroscedasticity in the data, meaning the variance of the errors is not constant across all data points.
- Constant Trace Least Squares Method (CTLSM): Explanation: The constant trace least squares method focuses on maintaining the trace (sum of diagonal elements) of the covariance matrix of the estimated parameters constant. It is particularly useful when dealing with parameter estimation in adaptive systems.
- Instrumental Variables Moving-Average (IVMA) Method: Explanation: The IVMA method is used to estimate parameters in a system when the available data is subject to noise or measurement errors. It employs instrumental variables to mitigate the effects of these errors on parameter estimation.
- Instrumental Variables Output-Error (IVOR) Method: Explanation: The IVOR method is similar to IVMA but specifically applies to output-error models, where the error in the system's output is taken into consideration during parameter estimation.
- Square-Root Information Variable (SRIV) Method: Explanation: The SRIV method is an extension of the instrumental variables approach. It is particularly useful for estimation of parameters in the presence of correlated noise or when dealing with ill-conditioned systems.
- Identification of Nonlinear Systems: Explanation: Identification of nonlinear systems involves estimating the behavior of systems with nonlinear dynamics. Various techniques, including neural networks, kernel methods, and polynomial models, can be employed to capture the complex relationships in such systems.