Control Theory: Basics and Hands-on

Guido Sassaroli

October 21, 2025

Contents

1 Introduction

The effective control of dynamic systems is essential in numerous engineering and industrial applications. This report provides a comprehensive overview of key control strategies, ranging from classical techniques such as PID control to advanced methodologies including MPC and Reinforcement Learning. Beginning with fundamental concepts, the report explores various control architectures. The discussion progresses to state-space approaches including pole placement and optimal control techniques like LQR and H_{inf} control. A significant portion is dedicated to MPC, highlighting both linear and nonlinear formulations, as well as robust variants designed to handle system uncertainties. Then, a brief introduction to data-driven control such as RL is presented. The report concludes a summary of practical tools used in control system analysis and implementation.

Through theoretical discussion and illustrative examples, this report aims to equip the reader with a clear understanding of the diverse control strategies.

2 Key concepts

This section introduces the fundamental concepts of control theory, serving as a foundation for the topics discussed in the subsequent chapters. By providing a clear overview of the main principles and terminology, it aims to establish a solid understanding that will support the reader's comprehension.

2.1 Stability

Stability refers to a system's ability to return to and remain near an equilibrium state when disturbed, or to maintain bounded outputs for bounded inputs. A system is considered stable if its output remains within a finite range despite variations in parameters or external disturbances.

In simpler terms, an equilibrium point x_e is said to be Lyapunov stable if all solutions that begin close to x_e remain close to it for all future time. Furthermore, if x_e is Lyapunov stable and all nearby solutions eventually converge to x_e, then x_e is considered asymptotically stable.

2.2 Controllability and Observability

In control systems, controllability refers to the ability to manipulate a system's state using external inputs, while observability refers to the ability to infer the system's internal state from its external outputs.

Controllability: Controllability refers to the ability to drive an initial system state to a desired final state within a finite time by choosing appropriate control inputs. In other words, a system is said to be completely controllable if it is possible to arbitrarily assign values to its states by proper choice of inputs.

Observability: In a state-space representation, a physical system is considered observable if the current state of the system can be determined from its outputs alone, regardless of the evolution of its state and control inputs. This means that, based solely on the output data (typically obtained from sensors), one can infer the internal behavior of the entire system. Conversely, if a system is not observable, there exist state trajectories that cannot be uniquely identified using only the output measurements. In such cases, different internal states may produce identical outputs, making it impossible to fully reconstruct the system's state from output data alone.

2.3 Continuous versus Discrete time

Continuous-time systems are those where signals and system dynamics evolve continuously over time. These systems are typically described using differential equations, where the behavior of the system is tracked at every instant. Continuous-time models are ideal for representing physical systems like mechanical or electrical processes where changes occur in an uninterrupted flow. They are commonly analyzed using tools such as Laplace transforms and are implemented in analog control systems or through digital controllers that approximate continuous behavior with high-speed sampling.

Discrete-time systems represent signals and dynamics at specific, separate time intervals. These systems are described using difference equations and are inherently digital, making them suitable for computer-based control and signal processing. Discrete-time models arise naturally when a continuous system is sampled by sensors or when digital controllers are used. While they

require techniques like the Z-transform for analysis, they offer advantages in implementation flexibility and robustness in the presence of noise and uncertainties. The choice between continuous and discrete time depends on the nature of the system and the design constraints of the control application.

2.4 Linear versus Nonlinear systems control

Linear control techniques deal with systems that can be modeled using linear equations, where the principle of superposition applies. These techniques are widely used because they offer analytical simplicity and well-established design tools such as root locus, Bode plots, and state-space analysis. Linear control is effective when the system dynamics are approximately linear or can be linearized around a specific operating point. Common linear controllers include Proportional-Integral-Derivative (PID) controllers and Linear Quadratic Regulators (LQR), which are efficient and robust for a wide range of practical applications.

Nonlinear control techniques are used for systems with dynamics that cannot be accurately captured by linear models, such as those with saturation, dead zones, or time-varying parameters. These systems do not obey the principle of superposition, making their analysis and control more complex. Nonlinear control methods, such as feedback linearization, sliding mode control, and Lyapunov-based design, are tailored to handle such complexities. While more challenging to implement and analyze, nonlinear control is essential for achieving desired performance and stability in systems where linear approximations are insufficient or fail altogether.

3 Information in control systems

3.1 Why signal processing matters in control?

Control engineering is all about driving a system toward a desired state. These systems can belong to very different domains: mechanical, electronic, energetic (of any sort), hydraulic, financial, or even biological. And the desired state can also vary a lot: do we want to improve performance? do we want to guarantee stability? do we want to implement some kind of autonomous reasoning?
What all these situations have in common, in all their possible combinations, is the need for *good* information.
Let's take a simple example: imagine you're assigned a school project where you have to program a pick-and-place robotic arm. The task is straightforward: pick up a ball and place it in a cup. To do so, you rely on information collected by a camera pointing at the robot's movements. With the sensor information (the camera), you design a reference trajectory based on the robot's kinematics, and then implement a PID controller to follow that reference. Easy, right? But here comes the twist: for budget reasons, you're constrained to a microcontroller with limited memory, so you have to use a low-resolution camera. Now your reference is based on a handful of pixels. When you check the camera feed, you discover that the signal is jittery, sensitive to shadows and light variations, and doesn't smoothly capture the robot's position. After you implement the controller, instead of a steady motion, the gripper "shakes" on its way to the target, slowing down the cycle time and often dropping the ball outside the cup.
Now consider a slightly different scenario: instead of relying on a camera, you have a force feedback sensor that tells you whether the ball has been grasped or not. Sounds promising, but here the problems are different. If the arm's electrical connections aren't perfectly insulated, or if the joints don't slide smoothly, the force signal may get corrupted. Disturbances like vibrations from the conveyor, jolts when the arm stops, or even electromagnetic interference can make the sensor "think" the ball has been grasped when it hasn't. As a result, the arm might prematurely stop closing the gripper, or misinterpret contact, either dropping or even crushing the object.
The key takeaway is that real-world settings are far from perfect. You can build a super-detailed simulation and design a perfectly tuned controller based on it, but once you move to reality there's always a good chance of running into unexpected phenomena. But that's part of the fun! It would be so boring if everything worked exactly as expected, wouldn't it? ;)
This is why we need signal processing before feeding information into our controllers: to filter out noise and disturbances that would otherwise ruin performance.
And there's more. In this chapter, we'll also look at a fascinating application of filters that goes beyond noise removal. Sometimes, to make your controller work, you need information about a state that you simply cannot measure directly. This opens the door to a very important branch of control theory: state estimation. More on that later.

3.2 Filters for raw signals

The very first and most basic thing you usually want to do with a sensor signal is filter out the high-frequency noise. But how do we actually do this in practice?
The idea is to design a transfer function that preserves the signal content in the frequency band of interest, while not introducing too much delay.

The first step is to look at the fastest pole of your system, call it λ. This pole corresponds to a frequency $f = \frac{|\lambda|}{2\pi}$. To keep the signal content up to f while attenuating high-frequency noise (let's

call its characteristic frequency f_n), you need to pick a cutoff frequency f_c that lies between f and f_n. From there, the corresponding time constant is $\tau = \frac{1}{2\pi f_c}$, and you can finally build your filter transfer function:

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

The key design choice here is the cutoff frequency f_c, which represents the trade-off between signal smoothness and delay. If you pick a very low f_c, your sensor signal will look very clean and smooth, but the introduced delay may degrade controller performance and even prevent you from achieving your control objective. On the other hand, a high f_c makes the system faster (closer to the original dynamics), but if it's too high, the filter will barely attenuate noise and become essentially useless.

PYTHON EXAMPLE

3.3 State observers and estimators overview

3.4 Luenberger observer

3.5 Kalman Filter

3.6 Other state estimators

3.7 Sensor models and measurement issues

3.8 Practical hands-on guidelines

4 PID

Proportional-Integral-Derivative (PID) control is the most widely used control algorithm in industrial applications, and its universal acceptance stems from two key factors: robust performance across a broad range of operating conditions and a straightforward implementation that simplifies its use by engineers.

As the name implies, the PID algorithm is built around three fundamental components—proportional, integral, and derivative terms. These parameters are adjusted to achieve an optimal system response.

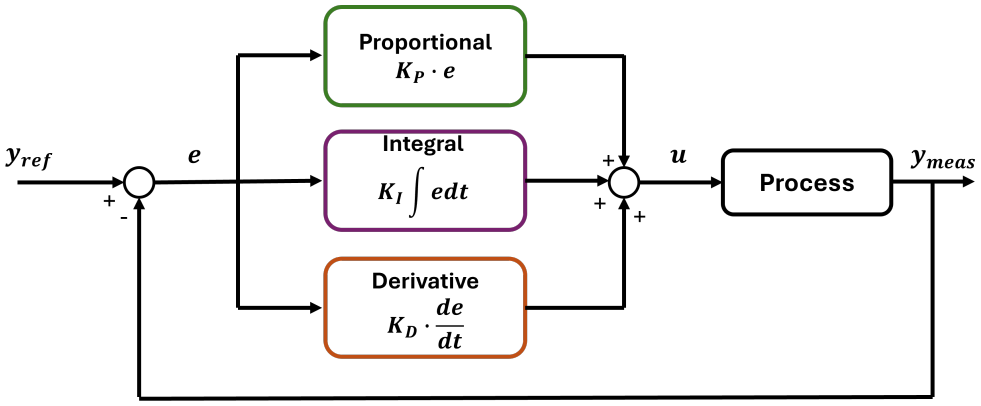

Figure 1: PID Control Scheme

The basic form of a PID controller can be described with this differential equation:

$$u = K_P\, e + K_I \int e\ \mathrm{d}t + K_D \frac{\mathrm{d}e}{\mathrm{d}t} \tag{1}$$

Where e describes the control error, i.e. the difference between setpoint and actual value of the controlled variable and u is the control action. The individual components are described with independent parameters K. A common alternative description form is:

$$u = K_P \left(e + \frac{1}{T_I} \int e\ \mathrm{d}t + T_D \frac{\mathrm{d}e}{\mathrm{d}t} \right) \tag{2}$$

Where the proportional component is described unchanged by proportional gain K_P, but the integral component is defined by the integration time T_I and the differential component by the derivative time T_D. Both newly introduced parameters have the dimension time.

Proportional Response: The proportional component depends only on the difference between the set point and the process variable. The proportional gain (K_P) determines the ratio of output response to the error signal. For instance, if the error term has a magnitude of 10, a proportional gain of 5 would produce a proportional response of 50.

Integral Response: The integral component sums the error term over time. The result is that even a small error term will cause the integral component to increase slowly. The integral response will continually increase over time unless the error is zero, so the effect is to drive the Steady-State error to zero. **Remark:** A phenomenon called integral windup results when integral action saturates a controller without the controller driving the error signal toward zero. In the next paragraph the anti-Windup PID will be presented.

Derivative Response: The derivative component causes the output to decrease if the process variable is increasing rapidly. The derivative response is proportional to the rate of change of the process variable.

4.1 PID Tuning

In general, increasing the proportional gain will increase the speed of the control system response. However, if the proportional gain is too large, the process variable will begin to oscillate. If K_P is increased further, the oscillations will become larger and the system will become unstable and may even oscillate out of control. A larger integral gain results in faster error correction, but also larger overshoots and oscillations. A smaller integral gain results in a more stable response, but also larger steady-state error and longer settling time. Increasing the derivative time (T_D) parameter will cause the control system to react more strongly to changes in the error term and will increase the speed of the overall control system response. Most practical control systems use very small derivative time, because the derivative response is highly sensitive to noise in the process variable signal. If the sensor feedback signal is noisy or if the control loop rate is too slow, the derivative response can make the control system unstable.

4.2 Anti Windup PID

In a standard PID controller, the Integral part of the controller keeps accumulating error over time. This is great for eliminating steady-state errors, but it can also cause a problem known as "integral windup". Integral windup happens when the controller output is saturated but the integral term keeps accumulating error as if nothing is wrong. This causes overshoot, slow recovery and unstable behavior. Basically, the controller is "trying too hard" and doesn't realize it's already doing all it can. Anti-windup is a technique used to limit or correct the integral term when the controller

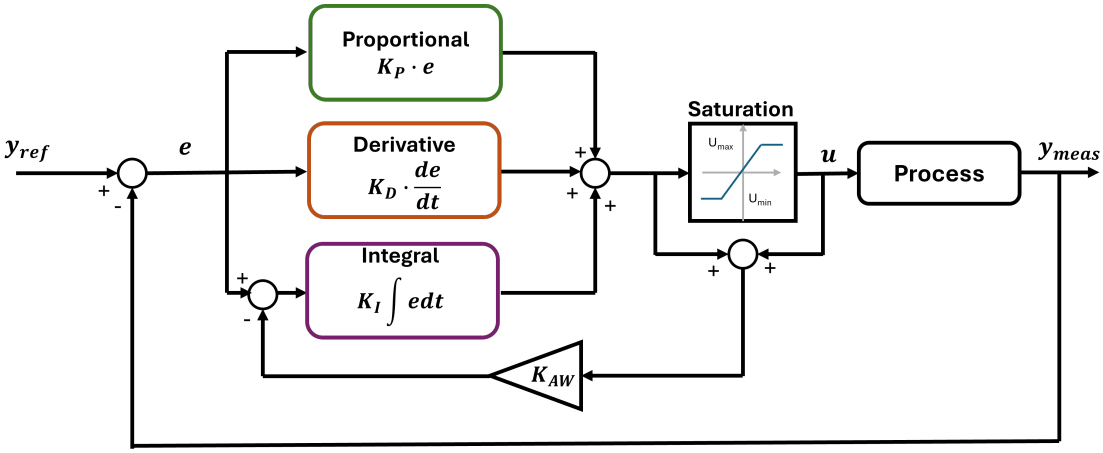

Figure 2: AW-PID Back-Calculation Control Scheme

output is saturated. It keeps the integral part from accumulating error that it can't act on. There are a few common methods, but a popular one is:

- Clamping Method (Conditional Integration): If the controller output is within limits: integrate normally. If the output is saturated: pause or limit the integral update.
- Back-Calculation Method: Adjust the integral term based on how far the controller output is from the desired (unsaturated) output.

Both methods aim to protect the controller from the adverse effects of windup, but they do so in different ways. Clamping works by halting integration under specific conditions, while back-calculation continuously adjusts the integrator in response to saturation. The choice between them often depends on the system dynamics, the frequency of saturation, and the desired smoothness of the control response.

4.3 Example

In the `PID.jpynb` file you can find the design of a PID to control the temperature of a room. The room is modeled considering the contribution of the heating from the heater and the cooling from the environment temperature. Then the temperature change can be modeled by:

$$\frac{dT}{dt} = k_{heat}P(t) - k_{cool}(T(t) - T_{env}) \tag{3}$$

Discretizing the differential equation we can obtain the following equation:

$$T_{next} = T_{current} + (k_{heat}P - k_{cool}(T - T_{env}))\Delta t \tag{4}$$

This is a good simplification, as the focus of the example is the implementation of the controller.

5 Loop Shaping

This section formalizes the notion of Loopshaping for linear control system design. The Loopshaping approach is inherently two-fold. First, we shape the open-loop transfer function $P(s)C(s)$, to meet performance and robustness specifications. Once this is done, then the compensator must be computed, from knowing the nominal product $P(s)C(s)$, and the nominal plant $P(s)$.

This holds for single-input, single-output systems, but the link to multi-variable control is not too difficult. In particular, absolute values of transfer functions are replaced with the maximum singular values of transfer matrices.

5.1 Example

The Python example `LoopShaping.jpynb` demonstrates the modeling and analysis of a basic control system using Loop Shaping. It defines a second-order plant, which represents a stable, overdamped system often seen in mechanical or electrical applications.

6 Main Control structures for energy systems

In this section some of the major control structures for energy systems are presented, remaining in the Linear Time-Invariant (LTI) context.

6.1 Feedforward compensation

The objective of this technique is to reduce the influence on the controlled variable y(t) of a measurable disturbance d(t) acting on the loop forward path. This can be achieved by computing K(s) so that Y(s)/D(s) is ideally zero.

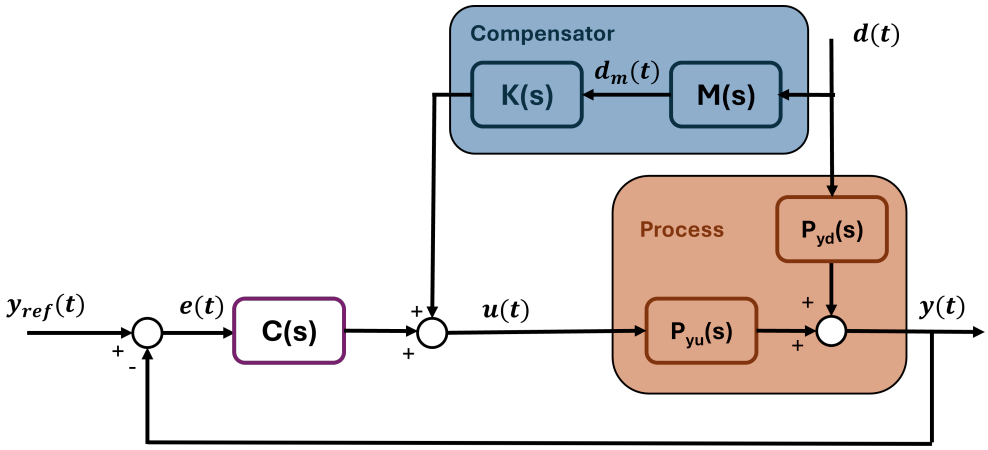

Figure 3: Feedforward Compensator Scheme

$$\frac{P_{yd}(s) + M(s)K(s)P_{yu(s)}}{1 + R(s)P(s)} = 0 \tag{5}$$

Then, in order to get this the ideal compensator is:

$$K_{ID}(s) = -\frac{P_{yd}(s)}{M(s)P_{yu}(s)} \tag{6}$$

This compensator is ideal because it has more zeros than poles, and it has poles in the right half plane (produce critical cancellations). So we can obtain the real compensator from the real one adding omitting zeros and/or adding poles. This compensation is valid up to a certain frequency, because at a certain point the $K(j\omega)$ starts to differ from $K_{id}(j\omega)$.

6.2 Cascade control

Cascade control is a method of control combining two feedback loops, with the output of one controller (the primary controller) adjusting the set-point of a second controller (the secondary controller). The cascade control is used to reduce the effect of a immeasurable disturbance $d_I(t)$ that affects some measurable intermediate process variable $y_I(t)$.

The inner (secondary) loop must be as fast as to hide both the dynamics of P_I and the effects of d_I to the outer (primary) loop.

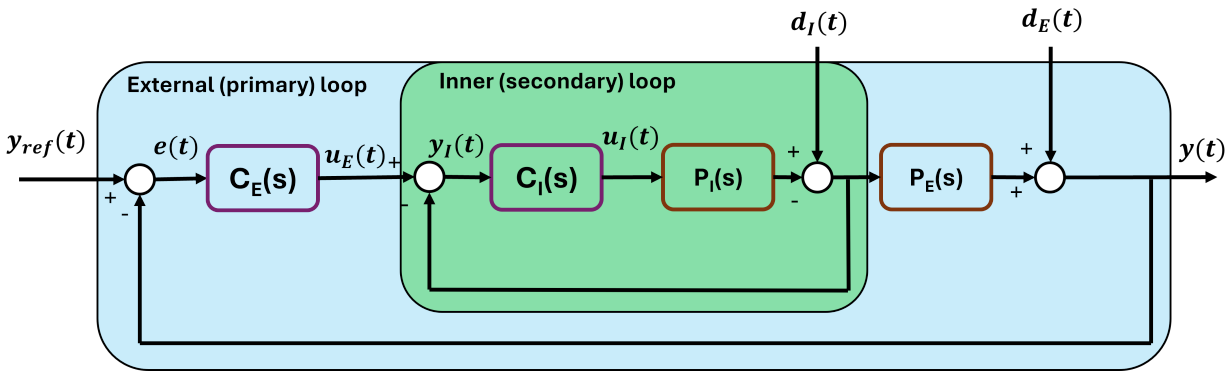

Figure 4: Cascade control scheme

6.3 Smith predictor

The Smith predictor is used when the process delay is large. With the Smith predictor is not necessary to sacrifice performance in order to obtain a certain stability degree. The Smith Predictor uses an internal model to predict the delay-free response $\hat{y}(t)$ of the process (e.g., what water temperature a given knob setting will deliver). It then compares this prediction $\hat{y}(t)$ with the desired setpoint $y_{ref}(t)$ to decide what adjustments are needed (control u).

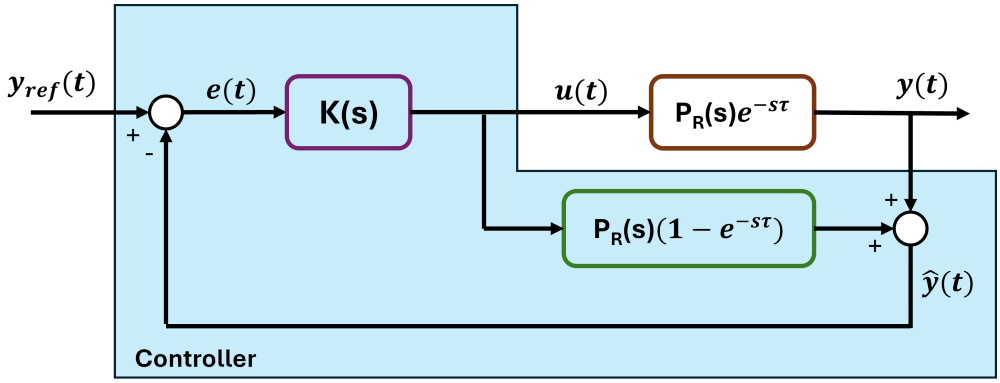

Figure 5: Smith Predictor Control Scheme

In the Scheme 5 we can see that:

$$\frac{\hat{Y}(s)}{U(s)} = P_R(s) \tag{7}$$

where the transfer function $P_R(s)$ is assumed rational. Block $K(s)$ can be synthetised with known methods, accounting only for the rational dynamics of the process.

Deploying the Smith Predictor scheme requires

- A model $P_R(s)$ of the process dynamics and an estimate τ of the process dead time.
- Adequate settings for the compensator dynamics ($K(s)$).

6.4 Split range

The purpose of this actuation control scheme is to make two actuators behave like a single one by having each of them act in a different range of control variable. A typical example is a tempera-

ture controller with one actuator for heating and one for cooling. This is easily achieved splitting the output into two different areas, one for the first actuator ($u_1 \in [0, 1]$) and one for the second actuator ($u_2 \in [-1, 0]$). The control action is defined in the following domain: $u \in [-1, 1]$. The simple setting is the following:

$$u_1 = \begin{cases} u & u \geq 0 \\ 0 & u < 0 \end{cases} \tag{8}$$

$$u_2 = \begin{cases} 0 & u \geq 0 \\ -u & u < 0 \end{cases} \tag{9}$$

Remark: Sometimes a dead zone is introduced around $u = 0$ to avoid switching in and out the two actuators too frequently.

6.5 Daisy-chaining

The purpose of this technique is to have several actuators activated in sequence when the previous one has reached its maximum. This logic is to start with the most energy-efficient actuator and the less efficient actuators intervene only if its necessary.

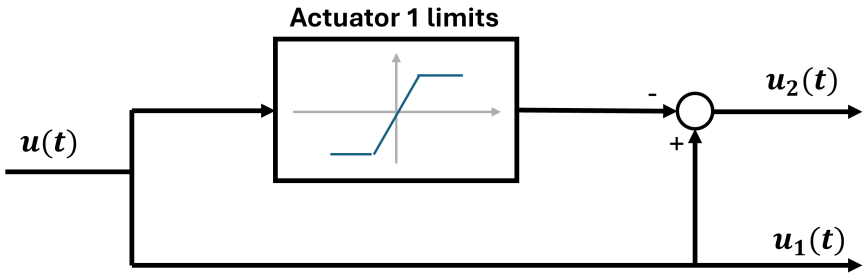

Figure 6: Daisy-Chain Scheme

Scheme 6 can be easily generalized to an arbitrary number of actuators.

Hints: Sometimes a dead zone is introduced around the transition to avoid switching in and out the two actuators too frequently.

7 Pole Placement

Pole placement is a method of calculating the optimum gain matrix used to assign closed-loop poles to specified locations, thereby ensuring system stability. Closed-loop pole locations have a direct impact on time response characteristics such as rise time, settling time, and transient oscillations. The following system describe the state-space system and the output equation:

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases} \tag{10}$$

with $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$. The control law is:

$$u = -Kx + \gamma \tag{11}$$

with $K \in \mathbb{R}^{m,n}$ and $\gamma \in \mathbb{R}^m$. The objective is to design a feedback matrix K such that the closed loop system

$$\dot{x} = (A - BK)x + B\gamma \tag{12}$$

has prescribed eigenvalues of the matrix $A - BK$.

The additional input γ does not play any role, but it can be useful to design external loops, while K is used for stabilization. Inner feedback used to stabilize, while R used for performance.

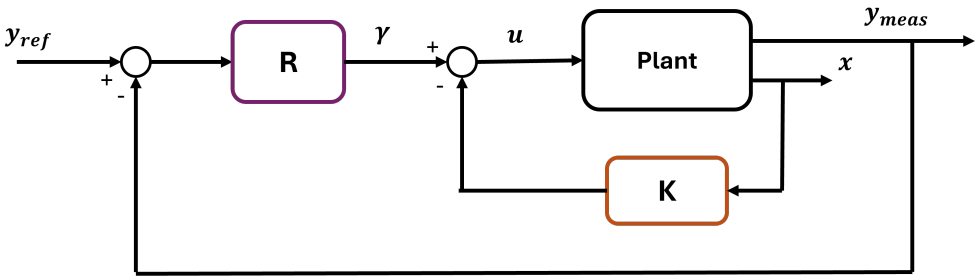

Figure 7: Pole-Placement scheme

Remark: We are assuming that the state is measurable (not realistic in many cases). This limitation can be practically addressed by using a state observer, which estimates the system's internal states based on available outputs.

7.1 Example

In the Python notebook `PolePlacement.jpynb` an implementation of Pole Placement control technique is implemented. In the Notebook a simple second-order plant in state-space form is designed. Then, two controllers with different pole placements are designed and the step responses are compared.

The plant in the example could represent a simple rotational mechanical system, like a DC motor with inertia and no damping. The states might represent angular velocity and angular position. The input u could be voltage applied to the motor. The output y is the angular position. So, the pole placement control the motor to reach a desired position.

8 $\mathbf{H}_{inf}$

In control theory, H_{inf} control offers a powerful framework for designing controllers that ensure robust performance in the presence of model uncertainties and external disturbances. Rather than optimizing for a specific nominal model, H_{inf} control focuses on minimizing the worst-case effect of disturbances across all frequencies. This is achieved by designing a controller that minimizes the H_{inf}-norm of the system's transfer function from disturbance inputs to performance outputs, effectively limiting the maximum energy amplification the system can experience. As a result, the controlled system is better equipped to handle a wide range of operating conditions and unforeseen perturbations. This robustness makes H_{inf} control particularly valuable in applications where performance and stability must be guaranteed despite uncertainties. The H_{inf} approach is inherently optimization-based, framing the controller design as a problem of achieving the best possible attenuation of disturbances, rather than simply following a reference trajectory or minimizing a specific cost function.

If you have a system described by matrices A,B,C,D and a disturbance w, and output z, you want to design a controller K so that the transfer function from w to z, call it T_{zw}, satisfies:

$$||T_{zw}||_{inf} < y \tag{13}$$

for the smallest possible y.

8.1 Example

Work in progress.

9 LQR

Linear Quadratic Regulator (LQR) is a feedback control algorithm that utilizes all state variables for optimal control design, where each state variable is multiplied by a gain and summed to generate a single actuation value. It is a stable control method based on specified performance weightings rather than eigenvalue placement.

Here just some hints in order to derive an LQR controller. Consider a linear time-invariant system in state-space form:

$$\dot{x} = Ax + Bu \tag{14}$$

and the infinite-horizon cost function given by:

$$J = \int_0^\infty \left[x^T \mathbf{Q} x + u^T \mathbf{R} u\right] dt, \quad \mathbf{Q} = \mathbf{Q}^T \geq \mathbf{0}, \mathbf{R} = \mathbf{R}^T > 0 \tag{15}$$

with Q and R symmetric and positive semi-definite. The goal is to find the optimal cost-to-go function $J^*(x)$ which satisfies the Hamilton–Jacobi–Bellman (HJB) equation:

$$\forall x, \quad 0 = \min_u \left[x^T \mathbf{Q} x + u^T \mathbf{R} u + \frac{\partial J^*}{\partial x} (\mathbf{A}x + Bu)\right] \tag{16}$$

The full process is skipped and it can be found in Chapter 8 of book (1).

The main results are that if the pair (A, B) is reachable and the pair (A, C_q) is observable, with C_q being the partition the matrix $Q = C_q^T C_q$, then:

1. The optimal control law is given by

$$u(t) = -\bar{K} x(t) \tag{17}$$

With

$$\bar{K} = R^{-1} B^T \bar{P} \tag{18}$$

Where $\bar{P}$ is the unique positive semidefinite solution of the stationary Riccati equation

$$0 = \bar{\mathbf{P}}\mathbf{A} + \mathbf{A}^T \bar{\mathbf{P}} - \bar{\mathbf{P}} B \mathbf{R}^{-1} B^T \bar{\mathbf{P}} + \mathbf{Q}. \tag{19}$$

2. The closed loop system $\dot{x}(t) = (A - B\bar{K})x(t)$ is asymptotically stable.

9.1 Example

The example `LQR.jpynb` demonstrates how to use a Linear Quadratic Regulator (LQR) to stabilize an inverted pendulum mounted on a cart. The system is linearized around the upright position, and the LQR controller is designed to keep the pendulum balanced while minimizing control effort. A linearized state-space model of the system is used and a force applied to the cart is used to move the cart.

10 MPC

Model Predictive Control (MPC) is a widely used control strategy in engineering and control theory, where it can be applied to a wide range of systems, including mechanical, electrical, and chemical processes. One key advantage of MPC is its ability to handle constraints on the system variables, such as limits on input and output variables, while still achieving optimal control performance. MPC also provides a framework for handling uncertainties in the system, through the use of robust optimization and adaptive control techniques.

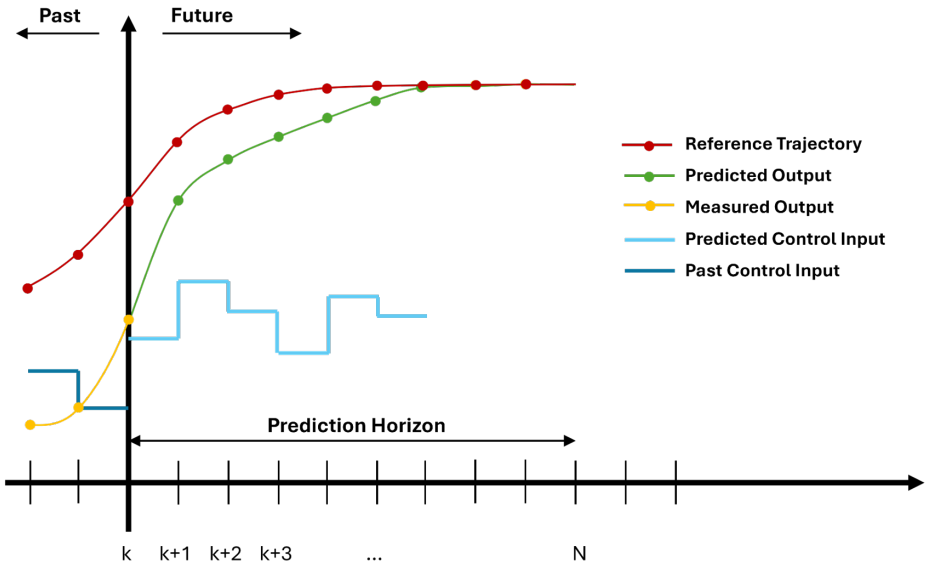

Figure 8: MPC Diagram

10.1 Linear MPC

The objective of MPC is to find the best control sequence over a future horizon of N steps. In particular Linear MPC uses linear process models and optimizes linear or quadratic performance objectives with linear constraints. The following equation describes a quadratic cost function:

$$l(x,u) = ||x - x_{ref}||_Q^2 + ||u||_R^2 \tag{20}$$

Eq. 20 represents the stage cost where

- x - states at current time
- x_{ref} - reference trajectory
- u - inputs
- Q - penalizing cost matrix for states
- R - penalizing cost matrix for input

The cost function can be written as a combination of running cost as shown in Equation 21:

$$J_N(x(t), u(\cdot|t)) = \sum_{k=0}^{N-1} l(x(k|t), u(k|t)) \tag{21}$$

Hence, the MPC is characterized by the solution of the following optimization problem:

$$V_N(x) = \min_u J_N(x, u) \tag{22}$$

Furthermore, the cost function must be minimized respecting the constraints on states and control actions:

$$\begin{aligned} x_{min} \leq y_k \leq x_{max} \\ u_{min} \leq u_k \leq u_{max} \end{aligned} \tag{23}$$

The Algorithm 1 explains how the MPC works. It receives as input the objective function J_N, the dynamic prediction model $f(x, \dot{x}, u)$, the horizon T and the initial guess $\hat{u}_{1:T}$, that is the previous solution of the MPC.

Algorithm 1 MPC algorithm

$u_{1:T} \leftarrow \hat{u}_{1:T}$
$x_{init} \leftarrow$ GetCurrentState
$x_{ref} \leftarrow$ GetReferenceTrajectory
$u_{1:T} \leftarrow$ SolveOptimizationProblem$(J_N, f, x_{init}, x_{ref}, T, \hat{u}_{1:T})$
$u \leftarrow$ First$(u_{1:T})$
ApllyInput(u)

Hints : A good procedure is to use the control sequence of the previous step to warm-start the algorithm and speed it up, that's why in line 4 of Algorithm 1 $u_{1:T}$ is given as input to the SolveOptimizationProblem function. By providing a good initial guess for the control input, the solver can start the optimization process from a point that is already close to the optimal solution. This can lead to faster convergence and improved control performance. In each iteration, the previous control input trajectory is used as an initial guess for the current iteration, taking advantage of the continuity and smoothness of the control signal.

10.2 Nonlinear MPC

Nonlinear model predictive control (NMPC) is the nonlinear counterpart of MPC and it has been applied to a wide range of complex systems. NMPC combines the predictive power of model-based control with the ability to handle nonlinear systems, making it a popular choice for many applications. The book by Rawlings, Mayne and Diehl (2) provides an excellent overview of NMPC, including its mathematical foundations and practical implementation. Overall, NMPC shows a wide-ranging applicability and a potential to solve complex control problems. While there are still challenges to be addressed, such as the computational demands of the optimization algorithm, the future of NMPC looks promising.

10.3 Robust MPC

Robust Model Predictive Control (RMPC) is a variant of MPC that is designed to be resistant to uncertainty and disturbance in the system. RMPC is often used in systems where the uncertainty and disturbance are difficult to model accurately, such as in systems with nonlinear dynamics or time-varying parameters.
There are several techniques that can be used to design a RMPC. A short state of the art on the most techniques will be presented in the following section.

10.3.1 Min-max MPC

It minimizes a cost function for the worst-case uncertainty realization. In particular it solves a min-max optimal control problem (3). The strong relationship between MPC and dynamic programming, where the former is an approximation of the latter, has resulted in comparable formulations for addressing RMPC problems. This method is really interesting from the theoretical point of view, especially when the uncertainty and disturbance in the system are bounded, but quite difficult to be applied in reality, in particular for computational complexity issues.

10.3.2 Scenario-based MPC

It is a probabilistic solution framework that assumes a finite number of possible values for uncertainties and models their realizations in a scenario tree (4), (5). This approach can guarantee performance in a probabilistic sense (satisfaction for most uncertainty instances) rather than a deterministic sense (satisfaction against all possible uncertainty outcomes). However, it can be computationally complex, especially when a large number of samples is required. Pippia et al. (6) propose a scenario-based MPC (SBMPC) controller with a nonlinear Modelica model, which provides a richer building description and can capture dynamics more accurately. The SBMPC controller considers multiple realizations of external disturbances and uses a statistically accurate model for scenario generation. Simulations show that the approach proposed by Pippia et al. outperforms standard controllers available in the literature in terms of building energy cost and comfort trade-offs.
The main advantage of scenario generation approaches is that they are applicable to wide classes of systems (linear, nonlinear) affected by general disturbances (additive, multiplicative, parametric, bounded or unbounded) with constraints of general type on the inputs and states.

10.3.3 Adaptive MPC

Performances of MPC is closely tied to the quality of the prediction problem. Adaptive MPC attempts to solve this problem taking inspiration from the adaptive control literature (7) to improve the model through a parameter update law based on execution error. This is a commonly used technique for linear models, but it can be difficult to apply to nonlinear models. Nonlinear AMPC is still a field of research and the solutions present in the literature often rely on unrealistic assumptions.

10.3.4 Tube-based MPC

Tube-based MPC (TBMPC) is a framework where a robust controller, designed offline, keeps the system within an invariant tube centered around a desired nominal trajectory, generated online. Bertsekas and Rhodes (8) were the first to present the idea of a tube and its role in robust contol. This framework was originally developed for linear systems. Mayne (9) extends this control technique to constrained nonlinear systems. However, nonlinear tube MPC is significantly more challenging than its linear counterpart. Additionally, the need fo two controllers, one for the generation of the nominal trajectory and one for the steering of the system towards this trajectory, makes the framework less scalable. Eventually, TBMPC usually exhibits overly conservative behavior.
Lopez, Slotine and How (10) try to overcome the conservativeness issue optimizing simultaneously the tube geometry and open-loop trajectory. The tube geometry dynamics is included to the nominal MPC optimization with minimal increase in computational complexity.

10.3.5 Constraint Tightening MPC

Another approach is Constraint Tightening MPC, which uses tightened constraints to achieve robustness. It is similar to tube MPC in that it uses a nominal model and tightens state constraints along the prediction horizon. The constraint tightening is based on the convergence rate of the dynamic system and the magnitude of the disturbance. This method captures the dispersion of trajectories caused by uncertainty and reduces the computational burden of calculating robust control invariant sets, which are necessary in tube MPC (11). Recent studies have used incremental stability (12) to obtain a lower bound on the convergence rate of the system, allowing for the tuning of a single parameter to obtain tightened constraints (13).

10.4 Examples

10.4.1 Example: Linear MPC

In the Python notebook `MPC.jpynb` an implementation of a linear MPC is presented. The system controlled is similar to a cart on a 1D track. So the controller is a trajectory tracker for this cart.

10.4.2 Example: Nonlinear MPC

Work in progress.

11 Reinfrocment Learning

Reinforcement Learning (RL) is emerging as a powerful framework for designing controllers in systems with complex, nonlinear, and uncertain dynamics. Unlike traditional model-based control methods, RL does not require an accurate system model and instead learns optimal control policies through interaction with the environment. This makes RL particularly attractive for real-world applications where system dynamics are difficult to model. For a deeper exploration of RL in general, I recommend the book by Sutton and Barto (14). For a focus on RL in control I recommend this article (15).

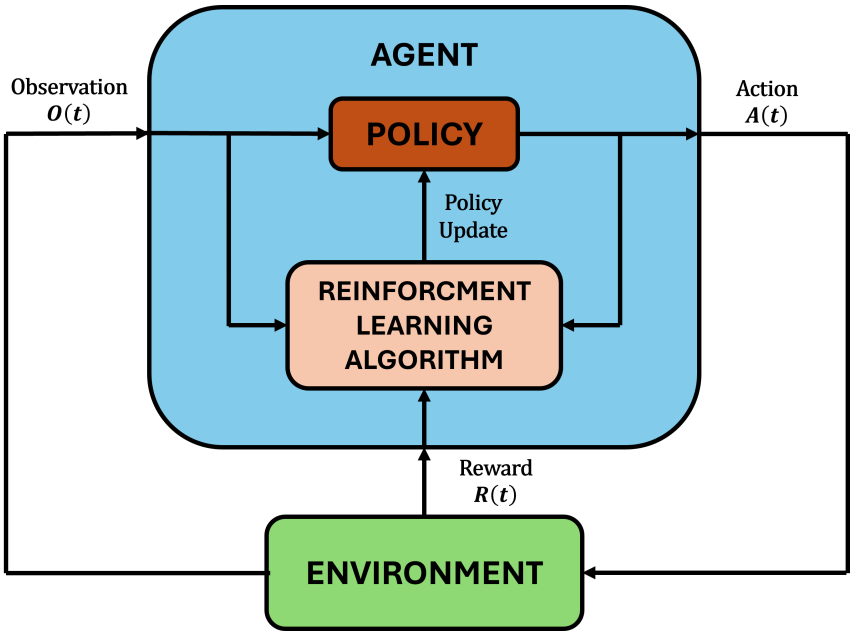

Figure 9: RL Scheme.

11.1 Example

Work in progress. In the meantime, consult the following repository: https://github.com/guidosassaroli/RL-cartpole.git. In this example a RL Agent is designed in order to control the Cart Pole environment (https://www.gymlibrary.dev/environments/classic_control/cart_pole/).

12 Useful Tools

In this section some useful control-related Python libraries and tools are presented.

Python Control Systems Library (16) is a Python package that implements basic operations for analysis and design of feedback control systems. It provides a range of tools for working with linear time-invariant (LTI) systems, including transfer functions, state-space models, and frequency response analysis. The library is widely used in academia for control engineering and is built on top of popular scientific libraries like NumPy and SciPy. It is a powerful and accessible tool for modeling, analyzing, and designing control systems. One of its main advantages is that it's open-source and free to use, making it ideal for students, educators, and researchers. However, it does come with some limitations. Compared to more mature tools like MATLAB's Control System Toolbox, it lacks some advanced features, such as robust control methods and support for nonlinear systems. Performance can also be a concern when dealing with large-scale or real-time systems, as Python generally doesn't match the execution speed of compiled environments.

Drake (17) is a toolbox started by the Robot Locomotion Group at the MIT Computer Science and Artificial Intelligence Lab (CSAIL). It is a collection of tools for analyzing the dynamics of robots and building control systems for them, with a heavy emphasis on optimization-based design/analysis.

cvxpy (18) is an open source Python-embedded modeling language for convex optimization problems. It lets you express your problem in a natural way that follows the math, rather than in the restrictive standard form required by solvers.

CasADi (19) is an open-source software tool for numerical optimization in general and optimal control (i.e. optimization involving differential equations) in particular. CasADi is not an "optimal control problem solver", that allows the user to enter an OCP and then gives the solution back. Instead, it tries to provide the user with a set of "building blocks" that can be used to implement general-purpose or specific-purpose OCP solvers efficiently with a modest programming effort.

Acados (20) is a software package for the efficient solution of optimal control and estimation problems. Acados is a complete rewrite of Acado, but unlike Acado it uses the symbolics of CasADi and the C code generation functionality. CasADi's focus is rapid prototyping of a range of different NMPC algorithms. The idea is to have an easy-to-use environment that can be used in both research and teaching. In contrast Acados implements SQP type solvers tailored to OCP structured NLPs, which aim to solve those problems very fast. The solution time of Acados for typical MPC problems is expected to be orders of magnitude faster compared to using IPOPT in CasADi.

do-mpc (21) is a open-source toolbox for MPC and moving horizon estimation (MHE). do-mpc enables the efficient formulation and solution of control and estimation problems for nonlinear systems, including tools to deal with uncertainty and time discretization. The modular structure of do-mpc contains simulation, estimation and control components that can be easily extended and combined to fit many different applications.

13 About the author

Hi, I'm Guido Sassaroli, I hold a degree in Automation and Control Engineering from Politecnico di Milano, and I completed my master's thesis at the Technical University of Munich, where I focused on Robust Nonlinear Model Predictive Control for Autonomous Driving. Right now I'm a researcher in the Department of Power Generation Technologies and Materials at Ricerca sul Sistema Energetico (RSE), Milan, Italy.
During my academic and professional journey, I've explored a wide range of control techniques. Still, I often found it challenging to connect theoretical concepts with real-world applications. To bridge that gap, I created this repository and accompanying document.
This project aims to provide a hands-on, accessible overview of fundamental control system techniques, supported by practical code examples and real-world applications. Although MATLAB/ Simulink is considered the go-to tool for control engineers, I've chosen to use Python and open-source libraries to make the content more approachable and freely available to everyone.
This repo is especially suited for students, educators, and enthusiasts who want to dive deeper into both classical and modern control strategies through interactive simulations and clean code.
This project, including both the document and the repository, is a work in progress. Feedback, suggestions, and collaboration are always welcome. Feel free to reach out!

References

[1] R. Tedrake, *Underactuated Robotics*. 2023.

[2] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2022.

[3] D. M. Raimondo, D. Limon, M. Lazar, L. Magni, and E. F. ndez Camacho, "Min-max model predictive control of nonlinear systems: A unifying overview on stability," *European Journal of Control*, vol. 15, no. 1, pp. 5–21, 2009.

[4] G. C. Calafiore and L. Fagiano, "Robust model predictive control via scenario optimization," vol. 58, pp. 219–224, jan 2013.

[5] M. C. Campi, S. Garatti, and M. Prandini, *Scenario Optimization for MPC*, pp. 445–463. Cham: Springer International Publishing, 2019.

[6] T. Pippia, J. Lago, R. De Coninck, and B. De Schutter, "Scenario-based nonlinear model predictive control for building heating systems," *Energy and Buildings*, vol. 247, 05 2021.

[7] K. J. Åström and B. Wittenmark, *Adaptive control*. 2013.

[8] D. Bertsekas and I. Rhodes, "On the minimax reachability of target sets and target tubes," *Automatica*, vol. 7, no. 2, pp. 233–247, 1971.

[9] D. Q. Mayne and E. C. Kerrigan, "Tube-based robust nonlinear model predictive control," *IFAC Proceedings Volumes*, vol. 40, no. 12, pp. 36–41, 2007.

[10] B. T. Lopez, J.-J. E. Slotine, and J. P. How, "Dynamic tube mpc for nonlinear systems," 2019.

[11] A. Richards and J. How, "Robust stable model predictive control with constraint tightening," pp. 6 pp.–, 2006.

[12] D. Angeli, "A lyapunov approach to incremental stability properties," *IEEE Transactions on Automatic Control*, vol. 47, no. 3, pp. 410–421, 2002.

[13] J. Köhler, M. A. Müller, and F. Allgöwer, "A novel constraint tightening approach for nonlinear robust model predictive control," pp. 728–734, 2018.

[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[15] L. Buşoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko, "Reinforcement learning for control: Performance, stability, and deep approximators," *Annual Reviews in Control*, vol. 46, pp. 8–28, 2018.

[16] S. Fuller, B. Greiner, J. Moore, R. Murray, R. van Paassen, and R. Yorke, "The python control systems library (python-control)," in *60th IEEE Conference on Decision and Control (CDC)*, pp. 4875–4881, IEEE, 2021.

[17] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019.

[18] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[19] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, 2018.

[20] Acados, "Acados: Fast and embedded solvers for nonlinear optimal control," 2019.

[21] F. Fiedler, B. Karg, L. Lüken, D. Brandner, F. B. M. Heinlein, and S. Lucia, "do-mpc: Towards fair nonlinear and robust model predictive control.," *Control Engineering Practice*, vol. 140:105676, 2023.