it means that we have the two Jacobians

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}, u) = \begin{bmatrix} 0 & 1 \\ -\dfrac{g}{l}\cos x_1 & 0 \end{bmatrix} \qquad \text{and} \qquad \frac{\partial \mathbf{f}}{\partial u}(\mathbf{x}, u) = \begin{bmatrix} 0 \\ \dfrac{1}{ml^2} \end{bmatrix} \tag{1.112}$$

Hence, for the first equilibrium point, we have

$$\mathbf{x}^* = \begin{bmatrix} \pi/2 \\ 0 \end{bmatrix} \Rightarrow \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x})\right|_{\mathbf{x}=\mathbf{x}^*, u=u^*} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{1.113}$$

so that the local dynamics around $\mathbf{x}^*$ are

$$\delta\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \delta\mathbf{x} + \begin{bmatrix} 0 \\ \dfrac{1}{ml^2} \end{bmatrix} \delta u \tag{1.114}$$

For the second equilibrium point, we have this time

$$\mathbf{x}^* = \begin{bmatrix} \pi/4 \\ 0 \end{bmatrix} \Rightarrow \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x})\right|_{\mathbf{x}=\mathbf{x}^*, u=u^*} = \begin{bmatrix} 0 & 1 \\ -\dfrac{\sqrt{2}}{2}\dfrac{g}{l} & 0 \end{bmatrix} \tag{1.115}$$

which gives the local linear dynamics

$$\delta\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 \\ -\dfrac{\sqrt{2}}{2}\dfrac{g}{l} & 0 \end{bmatrix} \delta\mathbf{x} + \begin{bmatrix} 0 \\ \dfrac{1}{ml^2} \end{bmatrix} \delta u \tag{1.116}$$

□

1.6 Input/Output representations

Early control techniques and what is now called classical control engineering considered mostly linear time-invariant systems, and in this context, the so-called Transfer Functions were widely used. Simply put, a Transfer Function (TF) is an input/output framework which represents how an input u is "transformed" into output y. A Transfer Function is expressed in the Laplace domain using the complex variable s (with $s \in \mathbb{C}$). Let us recall a few results related to Transfer Functions in this section.

1.6.1 From ODEs to TFs

Obtaining a Transfer Function from a linear ODE is actually quite simple. As an example, start with the second-order differential equation given by

$$\ddot{y}(t) + a_1\dot{y}(t) + a_0 y(t) = b_1\dot{u}(t) + b_0 u(t) \tag{1.117}$$

where we assume to have *zero initial conditions*. Then, recalling that, if $y(t)$ in the time domain gives $Y(s)$ in the Laplace domain, then the first time-derivative $\dot{y}(t)$ gives $sY(s)$ in the Laplace domain, while $\ddot{y}(t)$ gives in turn $s^2Y(s)$ and so on for higher-order derivatives. Hence, equation (1.117) is transformed into

$$s^2Y(s) + a_1 sY(s) + a_0 Y(s) = b_1 sU(s) + b_0 U(s). \tag{1.118}$$

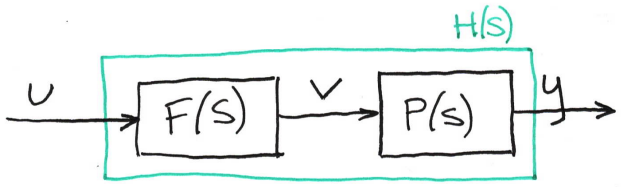

Figure 1.20: Two linear systems in cascade

After a simple factorization, we get

$$(s^2 + a_1 s + a_0)Y(s) = (b_1 s + b_0)U(s) \tag{1.119}$$

so that we have

$$Y(s) = \frac{b_1 s + b_0}{s^2 + a_1 s + a_0} U(s) = P(s)U(s) \tag{1.120}$$

where $P(s)$ is the Transfer Function of system/plant (1.117).
For more general linear systems, we have therefore

$$\frac{Y(s)}{U(s)} = \frac{b_m s^m + \ldots + b_1 s + b_0}{s^n + \ldots + a_1 s + a_0} = \frac{n_p(s)}{d_p(s)} \tag{1.121}$$

where the numerator $n_p(s) = b_m s^m + \ldots + b_1 s + b_0$ is a polynomial whose roots are called the *zeros* of Transfer Function $P(s)$, while the roots of polynomial denominator $d_p(s) = s^n + \ldots + a_1 s + a_0$ are called the *poles* of $P(s)$.

1.6.2 A simple stability criterion

The main advantage of Transfer Functions over linear ODEs is that TFs are algebraic equations instead of differential equations (as in ODEs). And the former are generally easier to manipulate and solve. Hence, the Laplace domain and TFs are useful for a wide range of computations and results, including assessing whether a system is stable or not, as the following simple result shows.

Stability criterion: A system described by Transfer Function $P(s)$ is said to be stable if and only if the real part of each pole (i.e. each root of denominator $d_p(s)$) is strictly negative[2]. □

1.6.3 Combining Transfer Functions

Combining two (or more) TFs gives yet another Transfer Function. But let us see more concretely in the following couple of examples.

[2]We also typically say "in the left half-plane", where plane is meant for complex plane.

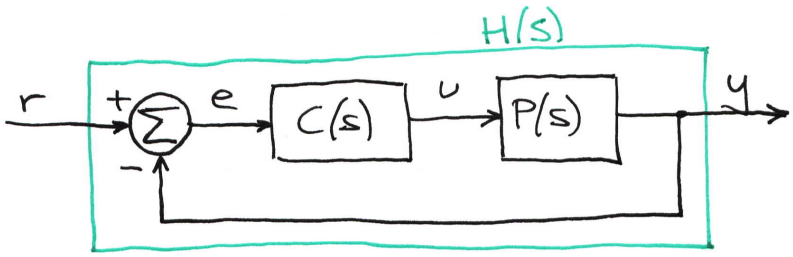

Figure 1.21: Controller and plant in feedback loop with unitary gain

Example: cascade of two systems

Consider the systems connected in cascade as illustrated in Figure 1.20, whose input/output representation is

$$V(s) = F(s)U(s) \tag{1.122}$$

for the first (sub)system with TF $F(s)$ (say a filter) and

$$Y(s) = P(s)V(s) \tag{1.123}$$

for the second (sub)system, with TF $P(s)$ (say the considered plant). Combining now (1.122) and (1.123), we get the IO relation

$$Y(s) = P(s)F(s)U(s) \tag{1.124}$$

where $H(s) = P(s)F(s)$ is the Transfer Function of the overall system. □

Example: feedback controller with unitary gain

Consider now a plant regulated by a controller with unitary feedback as illustrated in Figure 1.21, and where variable $e(t)$ is called the error and $r(t)$ is the reference that we want the output $y(t)$ to follow. Hence error signal $e(t)$ is defined as $e(t) := r(t) - y(t)$.
Using the results of the previous example and taking $E(s)$ as the Laplace transform of $e(t)$, we have

$$Y(s) = P(s)C(s)E(s), \tag{1.125}$$

while the very definition of the error variable $e(t)$ gives

$$E(s) = R(s) - Y(s). \tag{1.126}$$

Combine now (1.125) and (1.126), we get

$$Y(s) = P(s)C(s)\left[R(s) - Y(s)\right]. \tag{1.127}$$

Isolating then $Y(s)$, we get

$$\left[1 + P(s)C(s)\right]Y(s) = P(s)C(s)R(s), \tag{1.128}$$

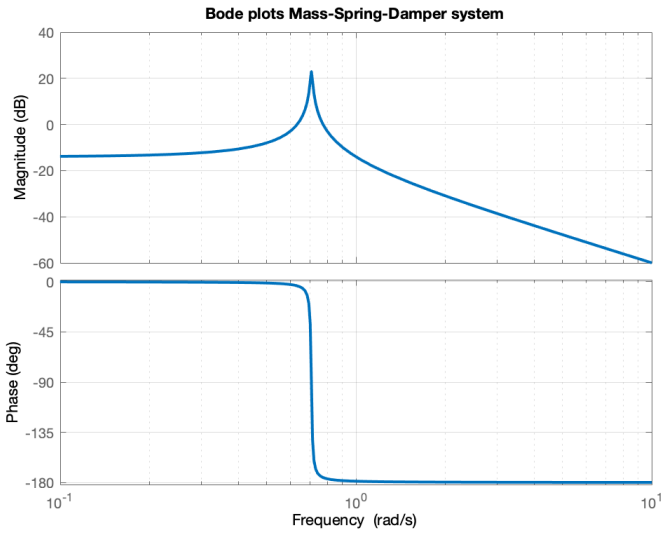

Figure 1.22: Bode plots of a "resonant" Mass-Spring-Damper system

which gives the new Transfer Function

$$H(s) = \frac{Y(s)}{R(s)} = \frac{P(s)C(s)}{1 + P(s)C(s)}. \tag{1.129}$$

In order to check whether $H(s)$ is stable or not, rewrite (1.129) in terms of numerators and denumerators of both the plant and controller:

$$H(s) = \frac{Y(s)}{R(s)} = \frac{\dfrac{n_p(s)n_c(s)}{d_p(s)d_c(s)}}{1 + \dfrac{n_p(s)n_c(s)}{d_p(s)d_c(s)}} = \frac{\dfrac{n_p(s)n_c(s)}{d_p(s)d_c(s)}}{\dfrac{d_p(s)d_c(s) + n_p(s)n_c(s)}{d_p(s)d_c(s)}} \tag{1.130}$$

which simplifies to

$$H(s) = \frac{n_p(s)n_c(s)}{d_p(s)d_c(s) + n_p(s)n_c(s)}. \tag{1.131}$$

Hence, $H(s)$ is stable if and only if the roots of $d_p(s)d_c(s) + n_p(s)n_c(s)$ are in the left half-plane.

1.6.4 Bode plots

A fundamental aspects of linear time-invariant systems, is that they preserve the nature of sinusoidal signals. Indeed, applying an input signal

$$u(t) = \sin(\omega t) \tag{1.132}$$

to the system leads to the output signal

$$y(t) = A\sin(\omega t + \varphi) \tag{1.133}$$

where both the amplitude A and the phase φ depend on angular frequency ω! Hence, one can perform a frequency analysis of a linear system by sweeping a range of sine waves at the input and see how they are amplified or reduced in amplitude (through A) and shifted in time (through φ). In this case, a dynamical system with input and output is seen here as a filter: what frequencies are attenuated, what frequencies are amplified, etc.
The so-called Bode plots are just that: two different plots where a TF $P(s)$ is evaluated at $s = j\omega$. One is called the magnitude plot and is defined by

$$A = |P(s = j\omega)| \ , \ \omega \in \mathbb{R}^+, \tag{1.134}$$

while the phase plot can be simply obtained by the expression

$$\varphi = arg(P(s = j\omega)) \ , \ \omega \in \mathbb{R}^+. \tag{1.135}$$

The phase plot is represented on a semi-log scale, with the log scale being on the angular frequencies, while the magnitude plot is represented on a log-log scale, i.e. we compute, instead of (1.134), $20 \log_{10} |P(s = j\omega)|$, and where the units are decibels.
In Matlab, the Bode plot are obtained readily by the command `bode`. In figure 1.22 are given the Bode plots of an MSD system for $m = 10$, $d = 0.1$ and $k = 5$.

1.6.5 Zero-frequency gain

The zero-frequency gain, also called Direct Current gain, is simply the value of the considered Transfer Function $P(s)$ when $s = 0$ or $\omega = 0$. In control systems, the zero-frequency gain is an important notion, particularly when the control objective is to stabilize the system around a constant reference. Indeed, the zero-frequency gain indicates how a constant level on the input u is amplified/reduced on the output y.

Example: MSD system

Let us come back to our (in)famous MSD system, whose differential equation is

$$m\ddot{y}(t) + d\dot{y}(t) + ky(t) = u(t) \tag{1.136}$$

The corresponding TF can be obtained by starting with

$$ms^2Y(s) + dsY(s) + kY(s) = U(S), \tag{1.137}$$

which can be then factorized into

$$\left(ms^2 + ds + k\right) Y(s) = U(s) \tag{1.138}$$

so that we have the TF

$$\frac{Y(s)}{U(s)} = P(s) = \frac{1}{ms^2 + ds + k} = \frac{1/m}{s^2 + (d/m)s + k/m} \tag{1.139}$$

The zero-frequency gain is then $P(0) = 1/k$. □

Chapter 2

Feedback control

2.1 Introduction

Roughly speaking, feedback control can be seen as making corrections so that a system (or its output $y(t)$ or the state $\mathbf{x}(t)$) behaves as you want, i.e follow some desired behavior (follows a reference $r(t)$ or a desired trajectory $y_d(t)$, etc.). Ideally, this should happen even when the system is starting not where you thought it would be starting from (different initial conditions), even when "something" is pushing the system away (disturbances) or when the system turns out to not be exactly what you thought it would be (system uncertainties).
In this chapter, our primary focus will be on state-space control techniques, after a very simple discussion on stability. But before doing so, let recall a few basics on the most famous feedback controller of all time, the Proportional-Integral-Derivative controller.

2.2 A primer on PID control

Following our short reminder of Transfer Functions in the previous chapter, a PID controller can be seen as feedback controller with unitary feedback gain (see Figure 1.21), where the Transfer Function $C(s)$ is given by

$$C(s) = k_p + k_i/s + k_d s \tag{2.1}$$

or, expressed in the time domain, we have

$$u(t) = k_p e(t) + k_d \dot{e}(t) + k_i \int_0^t e(\tau) d\tau \tag{2.2}$$

where we recall that $e(t) := r(t) - y(t)$ (as illustrated in Figure 2.1).
While there indeed a lot of interesting things that can be said on the PID controller, let us just recall a few keypoints, starting with the role of each of the three terms.

Roles of the different terms:

- P term: the primary role of the P term is to actually stabilize the system, ie to "attach it somewhere so that it won't go away".

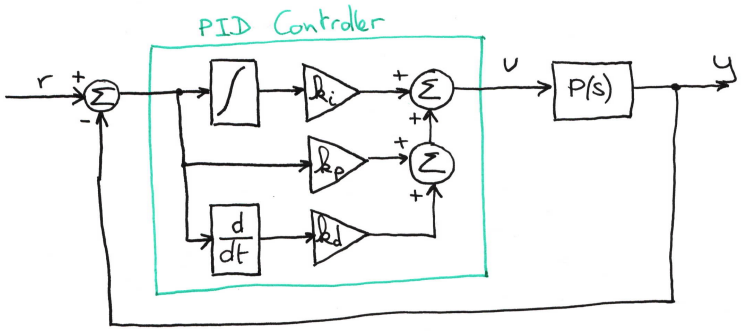

Figure 2.1: The Proportional-Integral-Derivative controller

- I term: this term uses the history of the difference between the output and the reference signal to infer the (typically) constant part of the control signal that will make the system output $y(t)$ stabilize right on $r(t)$, as well as compensating for constant to slowly-varying disturbances acting on the system.
- D term: finally, the D term helps to smooth things out, roughly speaking by adding damping in the system, so that the convergence of output $y(t)$ is not too oscillatory.

□

2.2.1 Tuning the controller: the Ziegler-Nichols rule

Tuning a PID controller can either be quite straightforward or a bit more delicate depending on a given plant, the environment where it evolves (possible disturbances, noise), as well as any specific desired operating conditions (eg. energy savings, wear and tear of actuator, etc.). Over the years, many guidelines were proposed to help a user to tune a PID controller on a specific plant. Amont them, the Ziegler-Nichols method is probably the most well-known. How it works can be summarized through the several few points given below.

- Start by putting all gains to 0.
- Change k_p until you can make the output y oscillate. When this happens, note the value of k_p and put it in k_c. Note then the period of oscillation T_c.
- Set the gain of your PID controller to

$$k_p = 0.6k_c, \quad k_i = \frac{k_p}{0.5T_c}, \quad k_d = k_p 0.125 T_c \tag{2.3}$$

Note that, even if this method works for a large class of systems, it does not do so for all systems. Furthermore, as mentioned above, these are just guidelines, and as such, they can be used to get a first rough tuning of the PID controller.

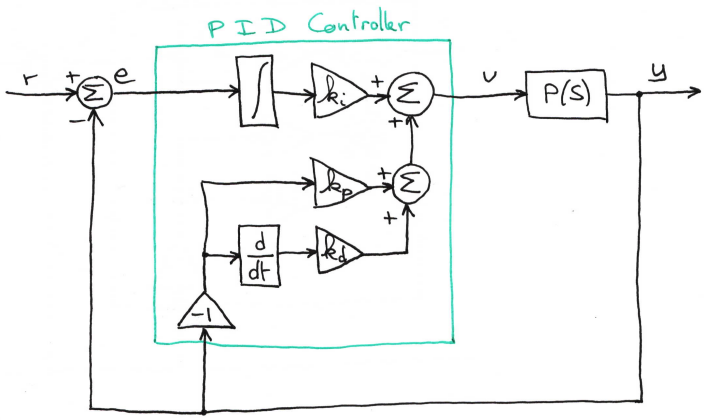

Figure 2.2: A modification of the PID controller

It is usually possible, if not recommended, to tune the different terms further manually to get better results.

2.2.2 Two implementation tips and tricks

Expression (2.1) (and its time-domain counterpart (2.2)) is, in a sense, the ideal or canonical way to express the essence of a PID controller. Indeed, the names of the different terms translate directly into the corresponding mathematical expression. However, direct implementation of (2.1) or (2.2) can, in practise, lead to several issues, some of which might be important depending on the application. While the literature is full of ways to solve these issues, let us see for now two "tips and tricks" that are frequently used in PID control.

Derivative term

As we have already discussed when making block diagrams out of differential equations, computing the derivative of a signal could be an issue because of added noise. It is especially relevant in the present case for PID control, as the derivative term implies the derivative of the error signal $e(r)$. The error containing the measurement noise-corrupted output $y(t)$, calculating $\dot{e}(t)$ directly will inject amplified noise into the closed-loop.
A simple way to remedy that is to replace the pure derivative term by a simple derivative filter, whose role is to remove the high-frequency component of the noise. Hence, expressed in the Laplace domain, the derivative term being given as

$$k_d s E(s) \tag{2.4}$$

is then changed into a pre-filtered version of $E(s)$, i.e. we have

$$k_d s \frac{1/T_f}{s + 1/T_f} E(s) \tag{2.5}$$

with

$$\frac{1/T_f}{s + 1/T_f} \tag{2.6}$$

being a simple low-pass filter.

"Jumps" in the reference signal $r(t)$

The reference signal $r(t)$ is typically set by the user. In case he/she suddenly change his mind about which constant reference should be followed by the system, we could therefore have a step input signal representing this change. The problem is that such a jump on signal $r(t)$ creates another jump at the output of proportional term k_p, hence in the contol input $u(t)$ sent to the actuator of the plant (see path followed by $r(t)$ to $u(t)$ in Figure 2.1), thus "shaking" it quite abruptly. This might have some bad effect depending on the considered plant. Even worse, the derivative term k_d implies a differentiation of the step signal, leading to an impulse being applied to the actuator , which is clearly not suitable for many mechanical systems, for example.
While we could also avoid that using a low-pass filter, as we have seen before, another trick is simply to pass the reference signal $r(t)$ only to the integrator, as seen in Figure 2.2. In this case, only the output $y(t)$ is passed to both the proportional and derivative term. Mathematically, we hence replace (2.2) by

$$u(t) = k_p(-y(t)) + k_d(-\dot{y}(t)) + k_i \int_0^t (r(\tau) - y(\tau))d\tau. \tag{2.7}$$

It can actually be proven that controller (2.7) is still able to stabilize a system around a constant reference signal r despite the absence of $e(t)$ in the proportional and derivative terms.