2.10 Model-Predictive Control (MPC)

2.10.1 From open-loop to feedback control

When we looked at the LQR controller (in section 2.5), we saw that we were trying to find a control input signal $\mathbf{u}$ over a time-horizon, such that a cost function J was minimized. Hence, for the discrete-time case, we want to find a sequence of N control inputs

$$\mathbf{u}(0), \mathbf{u}(1), \cdots, \mathbf{u}(N-1) \tag{2.110}$$

which will minimize the cost function

$$J = \sum_{k=0}^{N-1} \left[\mathbf{x}^T(k)\mathbf{Q}\mathbf{x}(k) + \mathbf{u}^T(k)\mathbf{R}\mathbf{u}(k)) \right]. \tag{2.111}$$

where N is called the horizon. Cost function (2.111) is minimized for systems represented by the discrete-time representation

$$\begin{cases} \mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k), & \mathbf{x}(0) = \mathbf{x}_0 \\ \mathbf{y}(k) = \mathbf{C}\mathbf{x}(k) \end{cases}. \tag{2.112}$$

In the above formulation, summarized by (2.110), (2.111) and (2.112), we have, in essence, an *open-loop control law*. Indeed, given model (2.112) starting at initial state $\mathbf{x}(0)$ (which we assume to be known or measured), the minimization of cost function/criterion (2.111) gives the sequence $\mathbf{u}(0), \mathbf{u}(1), \cdots, \mathbf{u}(N-1)$, which is then applied to system (2.112). It is open-loop because there is no feedback, ie the obtained state sequence $\mathbf{x}(1), \mathbf{x}(2), \cdots$ is not used to make corrections on $\mathbf{u}(0), \mathbf{u}(1), \cdots$!
A simple way to use the above strategy while adding a feedback mechanism to allow for corrections can be done as follows: start from $\mathbf{x}(0)$, obtain by cost minimization the input sequence $\mathbf{u}(0), \mathbf{u}(1), \cdots, \mathbf{u}(N-1)$, but apply *only* control input $\mathbf{u}(0)$ to system (2.112). Then, measure the state at time $k = 1$, ie $\mathbf{x}(1)$. Move now the horizon of the cost minimization problem from $\{0, \cdots, N-1\}$ to $\{1, \cdots, N\}$, and, starting this time from $\mathbf{x}(1)$, obtain by cost minimization the input sequence $\mathbf{u}(1), \mathbf{u}(2), \cdots, \mathbf{u}(N)$, and apply control input $\mathbf{u}(1)$ to system (2.112). Proceed similarly for subsequent iterations.
This moving-horizon control strategy, which is based on feedback information from the state $\mathbf{x}$ at each iteration, is called *Receding-Horizon Control* or *Model-Predictive Control*, MPC for short[1].
Hence, mathematically, our cost function (2.111), rewritten for every iteration/time k, will read

$$J(k) = \sum_{i=0}^{N-1} \left[\mathbf{x}^T(k+i)\mathbf{Q}\mathbf{x}(k+i) + \mathbf{u}^T(k+i)\mathbf{R}\mathbf{u}(k+i)) \right], \tag{2.113}$$

for which we will try to find, at each iteration k, the control sequence

$$\mathbf{u}(k+0), \mathbf{u}(k+1), \cdots, \mathbf{u}(k+N-1) \tag{2.114}$$

[1]The term "predictive" comes from looking at future control inputs and future states at each iteration.

which will minimize (2.113), while we will only apply the first term $\mathbf{u}(k+0)$ of sequence (2.114) to system (2.112) at each iteration.
Hence, at first glance, MPC can be seen as a slightly different way of calculating/expressing an LQR controller, with a finite horizon instead of a finite one.[2]
However, on the contrary to conventional LQR controllers, MPC also allows for the consideration of inequality constraints such as, for example,

$$\underline{\mathbf{u}} \leq \mathbf{u}(k) \leq \bar{\mathbf{u}}, \tag{2.115}$$

expression (2.115) being particularly useful to represent limits/saturations on actuators[3].

2.10.2 Quadratic Programming (QP)

Obtaining the sequence (2.114) that will minimize (2.113) under constraints such as (2.115) can generally *not* be computed explicitly. It can, however, be done numerically, using an optimization technique called *Quadratic Programming*, or QP for short.
Mathematically, quadratic programming consists in finding a vector $\mathbf{z}$ that will

$$\text{minimize } \frac{1}{2}\mathbf{z}^T\mathbf{H}\mathbf{z} + \mathbf{F}^T\mathbf{z}, \tag{2.116}$$

where $\mathbf{F}$ is a vector such that $\dim(\mathbf{z}) = \dim(\mathbf{F})$ and square matrix $\mathbf{H}$ is such that term $\mathbf{z}^T\mathbf{H}\mathbf{z}$ is a scalar.
To (2.116) can be adjoined constraints of several types, ie equality constraints of the type

$$\mathbf{G}\mathbf{z} = \mathbf{q} \tag{2.117}$$

and inequality constraints such as

$$\mathbf{W}\mathbf{z} \leq \mathbf{v} \tag{2.118}$$

or

$$\underline{\mathbf{z}} \leq \mathbf{z} \leq \bar{\mathbf{z}}, \tag{2.119}$$

with matrices $\mathbf{G}$, $\mathbf{W}$ and vectors $\mathbf{q}$, $\mathbf{v}$ are of appropriate dimensions.
In order to illustrate expressions (2.116)) to (2.118), and give a clearer intuition of what they represent, let us present a very simple example.

Example: a two-dimensional case

Consider the simple two-dimensional parabola equation represented by

$$P(\mathbf{z}) = (z_1 + 2)^2 + 10(z_2 + 3)^2 \tag{2.120}$$

and illustrated in figure 2.10. Let us now progressively rewrite equation (2.120)).
Indeed, we have

$$\begin{aligned} P(\mathbf{z}) &= z_1^2 + 10z_2^2 + 4z_1 + 60z_2 + 94 && (2.121)\\ &= \frac{1}{2}\mathbf{z}^T \begin{bmatrix} 2 & 0 \\ 0 & 20 \end{bmatrix} \mathbf{z} + \begin{bmatrix} 4 & 60 \end{bmatrix} \mathbf{z} + 94. && (2.122) \end{aligned}$$

[2]Recall indeed that for the LQR controller, the integration horizon is infinite (see equation (2.44))

[3]Expression (2.115) should be understood in a term-by-term sense, ie the i-th term of control input vector $\mathbf{u}(k)$ is bounded from above and below, ie $\underline{u}_i \leq u_i(k) \leq \bar{u}_i$

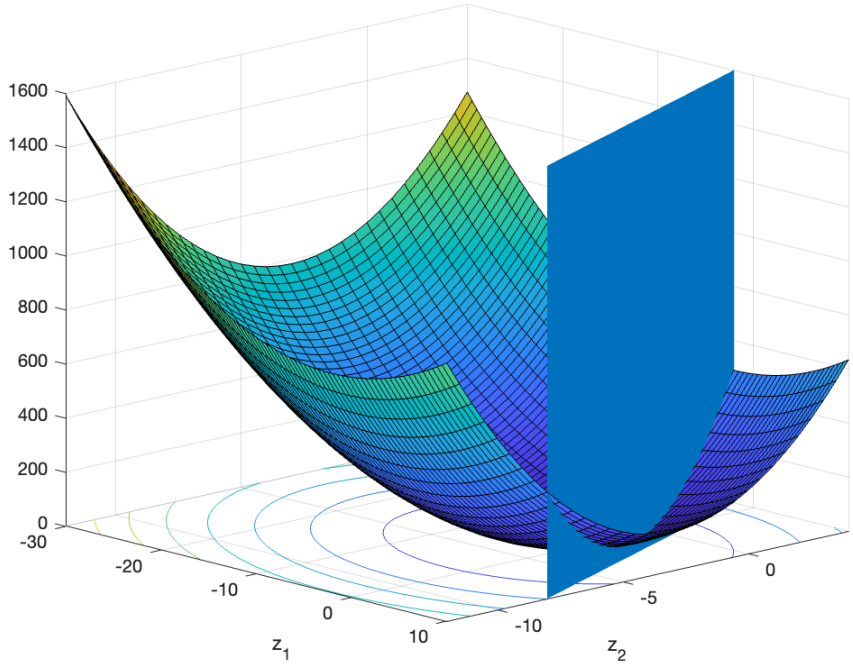

Figure 2.10: A parabola and a plane

Note, now, that expression (2.122)) is very close to (2.116)), and without the number 94, we would have actually obtained exactly (2.116)). As it turns out, finding the value of $\mathbf{z}$ for which $P(\mathbf{z})$ has its minimum does not depend on "how high" the parabola is, ie does not depend on number 94. Hence we can discard it and minimizing $P(\mathbf{z})$ can be done by solving the quadratic problem (2.116) with

$$\mathbf{H} = \begin{bmatrix} 2 & 0 \\ 0 & 20 \end{bmatrix} \quad \text{and} \quad \mathbf{F} = \begin{bmatrix} 4 \\ 60 \end{bmatrix}. \tag{2.123}$$

Assume now that we would like to minimize parabola (2.120) under the equality constraint

$$z_1 + z_2 = 2 \tag{2.124}$$

which is illustrated by the blue vertical plane in figure 2.10. Equality constraint can obviously be rewritten as

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \end{bmatrix} \mathbf{z} = 2, \tag{2.125}$$

which corresponds to expression (2.117) with $\mathbf{G} = \begin{bmatrix} 1 & 1 \end{bmatrix}$ and $q = 2$. Visually, equality constraint (2.124) means that the parabola function (2.120) will be minimized only on the part where it intersects with the vertical plane representing the equality constraint. Hence we are minimizing on a parabola of lower

dimension. Finally, assume now that, instead of equality constraint (2.124), we have inequality constraint

$$z_1 + z_2 \leq 2. \tag{2.126}$$

Then we have form (2.118) with $\mathbf{W} = \begin{bmatrix} 1 & 1 \end{bmatrix}$ and $v = 2$. Looking again at figure 2.10, it means visually that we are trying to find the value that minimizes the parabola for the domain situated behind the vertical plane (plane included). □

In Matlab, quadratic programming can simply be done with the command `quadprog`, while many options are also available in Python (see for example the package `cvxopt`).

2.10.3 From QP to linear MPC

If one wants to use the numerical tools available for QP (using Matlab or Python, for example), the MPC formulation summarized by expressions (2.113), (2.114), (2.115) should be put under the form (2.116) with the corresponding constraints. To do that, let us consider the simpler particular case where $N = 3$, and where, *at each iteration* k, we have[4]

$$\begin{aligned} J &= \sum_{i=0}^{3-1} \left[\mathbf{x}^T(i)\mathbf{Q}\mathbf{x}(i) + \mathbf{u}^T(i)\mathbf{R}\mathbf{u}(i))\right] \\ &= \mathbf{x}^T(0)\mathbf{Q}\mathbf{x}(0) + \mathbf{x}^T(1)\mathbf{Q}\mathbf{x}(1) + \mathbf{x}^T(2)\mathbf{Q}\mathbf{x}(2) \\ &\quad + \mathbf{u}^T(0)\mathbf{R}\mathbf{u}(0) + \mathbf{u}^T(1)\mathbf{R}\mathbf{u}(1) + \mathbf{u}^T(2)\mathbf{R}\mathbf{u}(2) \\ &= \begin{bmatrix} \mathbf{x}(0) \\ \mathbf{x}(1) \\ \mathbf{x}(2) \end{bmatrix}^T \begin{bmatrix} \mathbf{Q} & & \\ & \mathbf{Q} & \\ & & \mathbf{Q} \end{bmatrix} \begin{bmatrix} \mathbf{x}(0) \\ \mathbf{x}(1) \\ \mathbf{x}(2) \end{bmatrix} + \begin{bmatrix} \mathbf{u}(0) \\ \mathbf{u}(1) \\ \mathbf{u}(2) \end{bmatrix}^T \begin{bmatrix} \mathbf{R} & & \\ & \mathbf{R} & \\ & & \mathbf{R} \end{bmatrix} \begin{bmatrix} \mathbf{u}(0) \\ \mathbf{u}(1) \\ \mathbf{u}(2) \end{bmatrix} \end{aligned}$$

so that we get the more compact expression

$$J = \mathbf{X}^T \mathbb{Q} \mathbf{X} + \mathbf{U}^T \mathbb{R} \mathbf{U}, \tag{2.127}$$

with $\mathbf{X} := [\mathbf{x}^T(0), \mathbf{x}^T(1), \cdots, \mathbf{x}^T(N-1)]^T$ (with $N = 3$), with the matrices

$$\mathbb{Q} := \begin{bmatrix} \mathbf{Q} & & \\ & \mathbf{Q} & \\ & & \mathbf{Q} \end{bmatrix} \quad \text{and} \quad \mathbb{R} := \begin{bmatrix} \mathbf{R} & & \\ & \mathbf{R} & \\ & & \mathbf{R} \end{bmatrix}, \tag{2.128}$$

and where $\mathbf{U}$ is the vector containing the sequence of control inputs we want to find, ie

$$\mathbf{U} := [\mathbf{u}^T(0), \mathbf{u}^T(1), \cdots, \mathbf{u}^T(N-1)]^T, \tag{2.129}$$

again with $N = 3$.
Let us pause a moment and consider expression (2.127). In order to be able to use Quadratic Programming, we would like cost function expression (2.127) to resemble expression (2.116), with $\mathbf{U}$ replacing $\mathbf{z}$ as the unknown. While we are clearly taking that direction, we are not there yet. Indeed, while we know $\mathbf{x}_0$ as

[4]For simplicity and better readibility, we drop the dependency on k in the following mathematical development, keeping in mind that the subsequent calculations are valid for each iteration k.

the first element of vector $\mathbf{X}$, the latter is in general not directly known. However, note that, using state-space representation (2.112), we have

$$\begin{aligned}\mathbf{x}(0) &= \mathbf{x}(0)\\ \mathbf{x}(1) &= \mathbf{A}\mathbf{x}(0)+\mathbf{B}\mathbf{u}(0)\\ \mathbf{x}(2) &= \mathbf{A}\mathbf{x}(1)+\mathbf{B}\mathbf{u}(1)\\ &= \mathbf{A}\left[\mathbf{A}\mathbf{x}(0)+\mathbf{B}\mathbf{u}(0)\right]+\mathbf{B}\mathbf{u}(1)\\ &= \mathbf{A}^2\mathbf{x}(0)+\mathbf{A}\mathbf{B}\mathbf{u}(0)+\mathbf{B}\mathbf{u}(1).\end{aligned}$$

Rewriting the above in vectorial form, we get

$$\begin{bmatrix}\mathbf{x}(0)\\ \mathbf{x}(1)\\ \mathbf{x}(2)\end{bmatrix} = \begin{bmatrix}I_n\\ \mathbf{A}\\ \mathbf{A}^2\end{bmatrix}\mathbf{x}(0)+\begin{bmatrix}0 & 0 & 0\\ \mathbf{B} & 0 & 0\\ \mathbf{AB} & \mathbf{B} & 0\end{bmatrix}\begin{bmatrix}\mathbf{u}(0)\\ \mathbf{u}(1)\\ \mathbf{u}(2)\end{bmatrix} \tag{2.130}$$

so that we have

$$\mathbf{X} = \mathbb{A}\mathbf{x}(0)+\mathbb{B}\mathbf{U} \tag{2.131}$$

with new matrices $\mathbb{A}$ and $\mathbb{B}$ defined as

$$\mathbb{A} := \begin{bmatrix}I_n\\ \mathbf{A}\\ \mathbf{A}^2\end{bmatrix} \quad \text{and} \quad \mathbb{B} := \begin{bmatrix}0 & 0 & 0\\ \mathbf{B} & 0 & 0\\ \mathbf{AB} & \mathbf{B} & 0\end{bmatrix}, \tag{2.132}$$

ie vector $\mathbf{X}$ is now rewritten as (known) linear functions of $\mathbf{x}_0$ (known) and $\mathbf{U}$! Putting then (2.131) into (2.127), we have

$$\begin{aligned}J &= \left[\mathbb{A}\mathbf{x}(0)+\mathbb{B}\mathbf{U}\right]^T\mathbb{Q}\left[\mathbb{A}\mathbf{x}(0)+\mathbb{B}\mathbf{U}\right]+\mathbf{U}^T\mathbb{R}\mathbf{U}\\ &= \left[\mathbf{x}^T(0)\mathbb{A}^T+\mathbf{U}^T\mathbb{B}^T\right]\mathbb{Q}\left[\mathbb{A}\mathbf{x}(0)+\mathbb{B}\mathbf{U}\right]+\mathbf{U}^T\mathbb{R}\mathbf{U}\\ &= \mathbf{x}^T(0)\mathbb{A}^T\mathbb{Q}\mathbb{A}\mathbf{x}(0)+\mathbf{x}^T(0)\mathbb{A}^T\mathbb{Q}\mathbb{B}\mathbf{U}+\mathbf{U}^T\mathbb{B}^T\mathbb{Q}\mathbb{A}\mathbf{x}(0)\\ &\qquad +\mathbf{U}^T\mathbb{B}^T\mathbb{Q}\mathbb{B}\mathbf{U}+\mathbf{U}^T\mathbb{R}\mathbf{U}\end{aligned}$$

The last equality consisting of five (scalar) terms, combine, respectively, the second and third terms together[5], and the fourth and fifth terms together, to get

$$J = \mathbf{x}^T(0)\mathbb{A}^T\mathbb{Q}\mathbb{A}\mathbf{x}(0)+2\mathbf{x}^T(0)\mathbb{A}^T\mathbb{Q}\mathbb{B}\mathbf{U}+\mathbf{U}^T\left[\mathbb{B}^T\mathbb{Q}\mathbb{B}+\mathbb{R}\right]\mathbf{U}. \tag{2.133}$$

Cost function (2.134) should be minimized with respect to vector $\mathbf{U}$. Since term $\mathbf{x}^T(0)\mathbb{A}^T\mathbb{Q}\mathbb{A}\mathbf{x}(0)$ is known, then finding $\mathbf{U}$ that minimizes J is the same as finding $\mathbf{U}$ that minimizes

$$J' = \mathbf{U}^T\left[\mathbb{B}^T\mathbb{Q}\mathbb{B}+\mathbb{R}\right]\mathbf{U}+2\mathbf{x}^T(0)\mathbb{A}^T\mathbb{Q}\mathbb{B}\mathbf{U}, \tag{2.134}$$

which has the same form as quadratic programming problem (2.116), with $\mathbf{z} = \mathbf{U}$,

$$\mathbf{H} = 2\left[\mathbb{B}^T\mathbb{Q}\mathbb{B}+\mathbb{R}\right] \tag{2.135}$$

and

$$\mathbf{F} = 2\left[\mathbf{x}^T(0)\mathbb{A}^T\mathbb{Q}\mathbb{B}\right]^T. \tag{2.136}$$

[5]Recall that the transpose of a scalar is again a scalar.

There is obviously a lot more to MPC that what have been presented in these few derivations, and we have barely made a dent in scratching the surface. Indeed, and similarly to what we have seen for linear state-feedback, one can use MPC to stabilize around an equilibrium point, do tracking and even reject disturbances, one of the major interest of MPC is the ability to express constraints exemplified by actuator limits directly in the formulation of the control problem. This makes the MPC framework very powerful and MPC is a very active field of research which has found many applications out in the industry.

2.11 Linearization by state feedback: a small introduction to nonlinear state feedback

If linear state-feedback can work nicely for nonlinear systems, it does so sometimes only in a neighborhood around the chosen equilibrium point. However, as we will very briefly see in this section, the mechanisms behind linear state-feedback can be very easily extended to obtain *controllers that are nonlinear*. But before jumping into this, let us revisit linear state feedback with a slightly different perspective.

Start with the simple second order system (think of a MSD system with a mass of $m = 1$, for example) described by the state-space representation (in component form)

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -a_0 x_1 - a_1 x_2 + u \end{cases} \tag{2.137}$$

We have previously seen that, after using state-feedback

$$u = -\mathbf{K}\mathbf{x} = -k_1 x_1 - k_2 x_2, \tag{2.138}$$

we have the following closed-loop dynamics, itself a state-space representation

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -p_0 x_1 - p_1 x_2 \end{cases} \tag{2.139}$$

with $p_0 = a_0 + k_1$ and $p_1 = a_1 + k_2$, meaning that we could replace expression (2.138) with

$$u = -(-a_0 + p_0)x_1 - (-a_1 + p_1)x_2 \tag{2.140}$$

If we separate the system coefficients a_i from the closed-loop ones p_i, this last expression can be rewritten as

$$u = a_0 x_1 + a_1 x_2 - p_0 x_1 - p_1 x_2 \tag{2.141}$$

Putting now control law (2.141) back into system (2.137), we get the closed-loop dynamics

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -a_0 x_1 - a_1 x_2 + (\{a_0 x_1 + a_1 x_2\} + [-p_0 x_1 - p_1 x_2]) \end{cases} \tag{2.142}$$

where, in the second line of closed-loop system (2.142), the terms in parentheses are the terms clearly coming from controller expression (2.141). Note that we

have separated the terms of this control expression into two parts: a part in braces and a part in brackets.
As can be seen in the second line of (2.142), the part in braces is *exactly the opposite* of the system dynamics themselves. In control terminology, we say hence that this part of the control law *cancels* the system dynamics.
As for the part in brackets, it is simply there to obtain the new stable dynamics once the old system dynamics have been eliminated by the terms in braces. These terms in the brackets are usually called *stabilizing terms*. They correspond to our desired poles, the dynamics of the system in closed-loop.

Thus, the separation of these different kinds of term leads to consider two combined controllers: one in charge of cancelling the dynamics of the original plant or system, and one for stabilizing the newly-obtained dynamics, ie after cancellation. To see this, restart from scratch with system (2.137), whose dynamics we want to erase totally so that, after using our first feedback law, we would have the new dynamics

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = v \end{cases} \tag{2.143}$$

where v is a new control input, often called *virtual input*. To go from system (2.137) to (2.143), we used the "dynamics-cancelling" controller expression

$$u = a_0 x_1 + a_1 x_2 + v \tag{2.144}$$

Then, once we obtain the new dynamics (2.143), they are quite easy to stabilize since it is only a chain of two integrators. We can simply apply the stabilizing state-feedback term

$$v = -p_0 x_1 - p_1 x_2 \tag{2.145}$$

to obtain the stable closed-loop dynamics (2.139). Hence we have used a combination of two controllers for the stabilization of system (2.137) around the origin: a cascade of a controller cancelling the system dynamics with a controller stabilizing the newly-obtained dynamics.

More generally, if we take a system of order n described by the form

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_3 \\ \vdots \\ \dot{x}_n = -a_0 x_1 - a_1 x_2 - ... - a_{n-1} x_n + bu \end{cases} \tag{2.146}$$

where we assume, for simplicity, that $b = 1$, we can use a control law that will cancel the system dynamics

$$u = a_0 x_1 + a_1 x_2 + ... + a_{n-1} x_n + v \tag{2.147}$$

which will give new simple dynamics (ie a chain of integrators)

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_3 \\ \vdots \\ \dot{x}_n = v \end{cases} \tag{2.148}$$

that we can stabilize using the linear state-feedback controller

$$v = -p_0 x_1 - p_1 x_2 - ... - p_{n-1} x_n. \tag{2.149}$$

This basic idea of first cancelling a system dynamics to then stabilize the simpler system is what leads to the definition, for nonlinear systems, of *nonlinear* controllers. Let us see that on our (now famous) controlled-pendulum example.

Example: cancelling the nonlinear dynamics of the robot elbow

Let us start, as usual, with

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -\dfrac{g}{l}\sin(x_1) + \dfrac{1}{ml^2}u \end{cases} \tag{2.150}$$

Since this system is of second-order, we would like to define a controller that will cancel the system dynamics of the second line of (2.150) and obtain a system identical to (2.143). To do so, remark it implies that we want to define the virtual input as

$$v = -\frac{g}{l}\sin(x_1) + \frac{1}{ml^2}u \tag{2.151}$$

which means that, after isolating u, we have our "cancelling" controller

$$u = mgl\sin(x_1) + ml^2 v \tag{2.152}$$

Note that *this controller is nonlinear*. Because it allows to obtain linear system (2.143), expression (2.152) is therefore called *linearizing controller* or *feedback linearizing controller*, ie a controller that renders a system linear by feedback. Then, we just have to define a stabilizing controller identical to the one we have in (2.145). To summarize, putting (2.152) and (2.145) together in a global controller, we get the state-feedback control law

$$u = mgl\sin(x_1) + ml^2(-p_0 x_1 - p_1 x_2) \tag{2.153}$$

which is nonlinear. □

The above example can naturally be generalized to higher-order nonlinear system, even though this is quite beyond the scope of this course. However, the general principle or basic idea is still pretty much the same as the one presented in the few paragraphs above. One can also work on systems with several inputs. In order to give a flavour of that, let us see the following example.

Example: robot control

Consider a robot manipulator with n links modelled by the following nonlinear differential equation

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} \tag{2.154}$$

where $\mathbf{q}$ is the vector of joint angles. Symmetric and positive definite matrix $\mathbf{M}(\mathbf{q})$ represents the inertia of the system, while $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is the so-called Coriolis matrix, which is skew-symmetric. Vector $\mathbf{g}(\mathbf{q})$ accounts for the role played by

gravitational forces. The vector $\boldsymbol{\tau}$ represents the torques applied to each joint. Isolating the acceleration of vector $\mathbf{q}$, we get

$$\ddot{\mathbf{q}} = -\mathbf{M}^{-1}(\mathbf{q})\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{M}^{-1}(\mathbf{q})\mathbf{g}(\mathbf{q}) + \mathbf{M}^{-1}(\mathbf{q})\boldsymbol{\tau} \tag{2.155}$$

Defining then the velocity vector as $\mathbf{v} := \dot{\mathbf{q}}$, we can rewrite the second-order differential equation (2.154) in a state-space form as follows.

$$\begin{cases} \dot{\mathbf{q}} = \mathbf{v} \\ \dot{\mathbf{v}} = -\mathbf{M}^{-1}(\mathbf{q})\mathbf{C}(\mathbf{q}, \mathbf{v})\mathbf{v} - \mathbf{M}^{-1}(\mathbf{q})\mathbf{g}(\mathbf{q}) + \mathbf{M}^{-1}(\mathbf{q})\boldsymbol{\tau} \end{cases} \tag{2.156}$$

Notice that the above expression looks a lot like expression (2.150). Again, and similarly to the predious pendulum example, proceed to linearize (2.156) by defining the following virtual input (we will use another notation than v to avoid the confusion with the velocity vector).

$$\mathbf{a}_v = -\mathbf{M}^{-1}(\mathbf{q})\mathbf{C}(\mathbf{q}, \mathbf{v})\mathbf{v} - \mathbf{M}^{-1}(\mathbf{q})\mathbf{g}(\mathbf{q}) + \mathbf{M}^{-1}(\mathbf{q})\boldsymbol{\tau} \tag{2.157}$$

Isolating then $\boldsymbol{\tau}$, we obtain the feedback linearizing controller

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})\mathbf{a}_v + \mathbf{C}(\mathbf{q}, \mathbf{v})\mathbf{v} + \mathbf{g}(\mathbf{q}) \tag{2.158}$$

Putting this control law back into state-space representation (2.156), we get

$$\begin{cases} \dot{\mathbf{q}} = \mathbf{v} \\ \dot{\mathbf{v}} = \mathbf{a}_v \end{cases} \tag{2.159}$$

which is linear![6] From there, it should be easy to stabilize system (2.159) by linear state-feedback, ie we use the linear controller expression

$$\mathbf{a}_v = -\mathbf{K}_\mathbf{v}\mathbf{v} - \mathbf{K}_\mathbf{q}\mathbf{q}. \tag{2.160}$$

Putting now controllers (2.158) and (2.160) together, we have the overall nonlinear controller

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})\left\{-\mathbf{K}_\mathbf{v}\mathbf{v} - \mathbf{K}_\mathbf{q}\mathbf{q}\right\} + \mathbf{C}(\mathbf{q}, \mathbf{v})\mathbf{v} + \mathbf{g}(\mathbf{q}) \tag{2.161}$$

to stabilize system (2.154) (or (2.156)) around the origin. Note that making modifications to stabilize around another equilibrium point or around desired trajectories work similarly as what we have seen in the previous corresponding sections of this chapter. □

So we have been able to feedback-linearize systems, such as (2.150) or (2.156), for which the nonlinearity was at the same level, or in the same differential equation as the input. The general case where this does not occur is generally more complicated. However, using a nice trick involving a chosen output y, one can bypass this complication and feedback-linearize, provided that this output is well chosen ;-). This last statement is quite mysterious, so let us see on an example how this can be done.

[6] It should now be clear that virtual control input $\mathbf{a}_v$ represents an acceleration, hence the notation.

Example: feedback linearization through dynamics of an output

Consider the predator-predator dynamics expressed as

$$\begin{cases} \dot{x}_1 = x_1^2 + x_2 \\ \dot{x}_2 = x_1 + u \end{cases} . \tag{2.162}$$

Looking system at (2.162), we can see that, contrarily to the 2 previous examples, the first equation has nonlinearities (the term in x_1^2), while the control input is only present in the second equation. Therefore, a controller such as

$$u = -x_1 + v \tag{2.163}$$

will only result in the closed-loop dynamics

$$\begin{cases} \dot{x}_1 = x_1^2 + x_2 \\ \dot{x}_2 = v \end{cases} \tag{2.164}$$

i.e. we only transformed the 'bottom of the system' into an integrator and therefore failed to render the whole system linear by feedback. What can we do?
Let us first consider the output given by

$$y = x_1, \tag{2.165}$$

and then try to obtain a differential equation in y and its derivatives. Since system (2.162) is composed of 2 first-order differential equations, then we expect the resulting differential equation in y to be a second-order one[7]. Hence differentiate y to obtain, using the first line of (2.162),

$$\begin{aligned} \dot{y} &= \dot{x}_1 \\ &= x_1^2 + x_2 \\ &= y^2 + x_2 \end{aligned} \tag{2.166}$$

Isolating x_2 in the last expression above, we get

$$x_2 = \dot{y} - y^2 \tag{2.167}$$

whose derivative is

$$\dot{x}_2 = \ddot{y} - 2y\dot{y}, \tag{2.168}$$

so that the second line of (2.162) can be rewritten as

$$\ddot{y} - 2y\dot{y} = y + u, \tag{2.169}$$

or, isolating the highest-order derivative of this last equation,

$$\ddot{y} = 2y\dot{y} + y + u. \tag{2.170}$$

Let us now define a (nonlinear) state-space representation corresponding to differential equation (2.170). To do so, and in order to distinguish from the previous state $\mathbf{x} := [x_1, x_2]^T$, we define the new state noted $\mathbf{z}$ as

$$\mathbf{z} := \begin{bmatrix} y \\ \dot{y} \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}, \tag{2.171}$$

[7] similarly to our second-order MSD system corresponding to a state-space representation with the state being of dimension 2 (see chapter 1).

so that output dynamics (2.170) are equivalent to

$$\begin{cases} \dot{z}_1 = z_2 \\ \dot{z}_2 = 2z_1 z_2 + z_1 + u \end{cases} . \tag{2.172}$$

Looking now at new state-space representation (2.172), it is now way easier to linearize by feedback as all nonlinear terms are at the same level as the input. Hence, define the nonlinear controller

$$u = -2z_1 z_2 - z_1 + v \tag{2.173}$$

so that we get, after feedback-linearization, the linear dynamics

$$\begin{cases} \dot{z}_1 = z_2 \\ \dot{z}_2 = v \end{cases} . \tag{2.174}$$

□

The above example generalizes to more complicated systems. Note the importance in choosing the right output, ie the output that will result in an ODE whose order is the same as the dimension of the original system. While there is a systematic way to determine such output for SISO systems (this technique being beyond the scope of this course), this is not the case for systems with several inputs.