TÉCNICO LISBOA

Master in Electrical and Computer Engineering

2021/2022 – P4

Estimação e Controlo Preditivo Distribuído

Distributed Predictive Control and Estimation

Laboratory work

Picture source: SpaceX

Prepared by

João Miranda Lemos

Instituto Superior Técnico

Departamento de Engenharia Eletrotécnica e de Computadores

Área Científica de Sistemas, Decisão e Controlo

Ethics statement

When delivering the resolution of this work, the group of students that signs it guarantees, by this act, that the text and all the software and results delivered were entirely carried out by the elements of the group, with a significant participation of all of them, and that no part of the work or software and results presented were obtained from other persons or sources.

Objective

Practice with basic concepts of model predictive control (MPC). Solution of a control problem with MPC using a software package.

Underlying concepts

- Constrained optimization
- Receding horizon control
- Model predictive control
- State estimation

Laboratory guide organization

This laboratory guide is organised in four parts:

1. Basics on constrained optimization
2. Basics on receding horizon control
3. Basics on predictive control
4. MPC control design for the inverted pendulum

Hereafter, Pi identifies part i of the work to be done, as listed above.

Software required

To perform this work, the following software is required:

- Matlab, available through the IST campus license;
- SIMULINK, available through the IST campus license;
- Matlab Optimization Toolbox, available through the IST campus license;
- MPCtools 1.0, available at the Fénix page of the course

MPCtools is a set of Matlab functions that allow to simulate a MPC controller using Matlab and SIMULINK. MPCtools is a toolbox developed by The Department of Automatic Control, Lund Institute of Technology, Sweden, that is free to use for all purposes. Details on how to install and use MPCtools can be seen in the Reference Manual (see references). The information required to performed this work is described below in section 3.

Grades

The final grade of the laboratory is computed as

$$L=0.1\ G1 + 0.1\ G2 + 0.3\ G3 + 0.5\ G4$$

where Gi is the grade obtained in the part Pi of the work.

Appendix 1 lists the good practices that must be followed when writing the report. Failure to comply with these practices will imply a penalty on the grade. Appendix 1 can be used as a check list to verify the report and improve it.

Format of the report to deliver

The report to deliver must comply with the following format:

- The report can be written in either Portuguese or English. It is advised that you use the language that you master better. Poor language will be penalized.
- File format: pdf
- Each group of students delivers a report, signed by all the group members.
- Include in the 1st page:
    - The name of the curricular unit and the academic year (*ano letivo*)
    - Student number, name and email of all the students in the group
    - An ethical commitment to originality with the following text:
      The group of students identified above guarantees that the text of this report and all the software and results delivered were entirely carried out by the elements of the group, with a significant participation of all of them, and that no part of the work or the software and results presented was obtained from other people or sources.
- The answers must be given sequentially from page 2 of the report, indicating at the beginning of each one the respective number (P1, P2, …) in bold type.
- The maximum number of pages of the report to be delivered, including the cover page, is 15, with character font size 12, in LATEX or another word processor.
- Pages must be numbered, with the numbers preferably placed in the centre of the bottom line.

P1 – Basics on constrained optimization

Motivation

The key issue in MPC is that it is based on on-line minimization of a cost function to obtain the value of the control variable to apply to the plant. In this way, MPC allows

to enforce constraints on several variables, a feature that is unique among on-line design techniques (optimal control based on Pontryagin's Principle may also embed constraints, but this design technique is an off-line one).

Hence, it is important to get acquainted with algorithms and software for function minimization. This work provides an exercise on the use of popular function minimization Matlab tools. Although basic, this knowledge is very useful in writing our own tools for MPC, as well as in using MPC toolboxes.

Objectives

Get acquainted to the basics of numerical function minimization using Matlab Optimization Toolbox through the solution of simple two-dimensional problems, namely

- *fminunc*, for unconstrained function minimization
- *fmincon*, for constrained function minimization

In two-dimensional problems the function to minimize can be represented graphically either by using the Matlab functions

- *surf*, that draws a 3-dimensional plot of the function
- *contour*, that plots the level curves (the level curves of a function $y = f(x_1, x_2)$ are the curves defined by $f(x_1, x_2) = C$, for different values of the constant C).

Both *surf* and *contour* plot the function in a grid of points that can be defined using the function

- *meshgrid*

In addition to the example provided below, you should read the *help* and *doc* (in the Matlab command window type, for instance, *help fmincon* or *doc fmincon*) to get more information about these functions, as well as the syntax of their use.

A useful example

This example illustrates how to use the minimization software described above to compute unconstrained and constrained minima for a function. This problem is a trivial one, the aim being just to illustrate the use of the Matlab software.

The function to minimize is

$$f(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 1)^2.$$

It is obvious that this function has an unconstrained minimum for $x = (1, 1)$.

This unconstrained minimization problem is simply formulated as

$$\min_x f(x)$$

To use the Matlab solver *fminunc* to solve this problem, start by creating a Matlab function script with the name of the function to minimize. For instance, calling the function BasicFunction, you must create the script:

```
function f = BasicFunction(x)
f = (x(1)-1)^2 + (x(2)-1)^2;
end
```

To solve numerically the unconstrained optimization problem, define the initial guess of the optimal value of x (called x0 below) and run the following code

```
options = optimoptions('fminunc','Algorithm','quasi-newton');
xopt=fminunc(@BasicFunction,x0,options)
```

The first line defines the options of the solver (see Matlab help) and the second line calls *fminunc* to actually solve the problem. The name of the script that contains the definition of the function to minimize is given as the first argument of *fminunc* by the name of the script preceded by the symbol @.

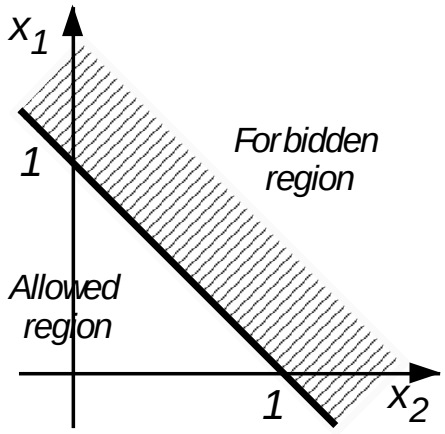

Figure P1-1. Basic optimization example: definition of a linear inequality constraint.

Consider now the introduction of an inequality constraint, such as

$$x_2 \leq 1 - x_1,$$

that corresponds to the allowed values below the straight line in figure P1-1. This constraint can be written in the form

$$Ax \leq B, \text{ with } A = [1 \quad 1] \text{ and } B = 1.$$

This form of the constraint is the standard one to be specified in the Matlab solver software function *fmincon*.

With the above constraint, the optimization problem is formulated as

$$\min_x f(x)$$
$$subject\ to\ Ax \leq B$$

The numerical solution of this constrained problem using *fmincon* is coded in Matlab by

```
Ac=[1 1];
Bc=1;
xoptconstr=fmincon(@BasicFunction,x0,Ac,Bc)
```

The first two lines define the constraint. The rest is similar to the use of *fminunc* described above. Of course, one must also define the initial guess of the optimal value of x (called x0).

💻 Work to perform

0) The Matlab scripts *BasicFunction.m* and *ProbBasic.m* are available in Fénix. These scripts allow to solve the two unconstrained, and constrained, minimization problems described above. They also plot a 3-D view of the function as well as its level curves. Study and run the script *ProbBasic.m* . You can use it as prototype to the other activities in this set of questions. In this question you don't have to write anything on your report.

1) Compute the unconstrained minimum of the Rosenbrock function, given by

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2.$$

Compute also the minimum that satisfies the constraint

$$x_1 \leq 0.5.$$

In both cases take as the initial estimate of the minimum

$$x_0 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

In the same plot, show the level lines of the function, the initial estimate (marked with "o"), the unconstrained minimum (marked with "x"), and the constrained minimum (marked with "*"). In another figure, plot the function.

2) Consider the function

$$f(x) = x_1^4 - 10x_1^2 + x_2^4 - 10x_2^2.$$

By using the previously described software tools for function plot and minimization, show that this function has several local minima. Consider one of the local minima (of your choice). Find an estimate of the boundary of its atraction basin, that is to say, of the set of points of the initialization points of the minimization algorithm such that *fminunc* converges to the minimum that you have considered. Sugestion: run *fminunc* with many initial diferente points. In some cases the result of *fminunc* is correct, while in other cases is not. Find the boundary of the set of initial points for which there is convergence to the minimum using the Matlab function *boundary* (see help). Explain how you select the initial points. There are several ways to perform this selection, some very simple, others more sophisticated; the objective is to achieve a good trade-of between computational load and accuracy of your estimate of the attraction basis boundary.

P2 – Basics on receding horizon control

Motivation

The receding horizon strategy is the key component of model predictive control (MPC). It consists of designing the control action such as to optimize it over an extended horizon, apply to the plant only the first control move of the optimized control sequence, and then repeated the process at the next discrete time instant, starting from the new state. It is the receding horizon strategy that allows to transform an open-loop optimal control action into a feedback control action.

When enlarging the optimization horizon, the initial control move approximates the one that would be obtained with an infinite horizon, which is stabilizing.

Objectives

The objective of the work to be done here is to compute the state feedback gain yield by the receding horizon strategy in a problem with linear dynamics, quadratic cost and without constraints, and to compare it with the infinite horizon gain computed using the solution of an equivalent linear quadratic problem with infinite horizon, that involves the solution of an algebraic Riccati equation.

[1] Cost optimization

Infinite horizon LQ problem

Consider the infinite horizon linear quadratic (LQ) optimal control problem that consists of minimizing the infinite horizon unconstrained quadratic cost

$$J_{LQ}(u) = \sum_{t=1}^{\infty} x^T(t)Qx(t) + u^T(t)Ru(t)$$

subject to

$$x(t+1) = Ax(t) + Bu(t).$$

The state x has dimension n and, in general, u has dimension m.

[1] This symbol means: *study this part carefully at home*.

The solution to this problem is obtained by solving, in order to the positive definite $n \times n$ matrix S, the discrete-time algebraic Riccati equation (ARE)

$$A^T SA - S - A^T SB(B^T SB + R)^{-1} B^T SA + Q = 0,$$

and then compute the optimal LQ gain by

$$K_{LQ} = (B^T SB + R)^{-1} B^T SA.$$

The optimal LQ control is given by the linear state feedback

$$u(t) = -K_{LQ} x(t).$$

The optimal LQ control exists if:

1. (A, B) is stabilizable
2. $R \succ 0$ (read $\succ$ positive definite) and $Q \succcurlyeq 0$ (read $\succcurlyeq$ positive semi-definite)
3. (Q, A) has no unobservable mode on the unit circle. If $Q = C^T C$, with C a vector, this condition is verified if (C, A) is observable.

Numerically, the discrete time LQ optimal gain can be computed using the MATLAB Control Systems Toolbox function *dlqr*. The syntax to call this function is

```
[KLQ,S,lambda] = dlqr(A,B,Q,R)
```

where KLQ is the optimal gain, S is the positive definite solution of the Riccati equation and lambda is the vector of eigenvalues of the closed-loop system dynamics $A - BK_{LQ}$.

Finite horizon, receding horizon control

Consider now the finite horizon unconstrained quadratic cost, defined over a horizon with H steps, given by

$$J_{RH}(u; t) = \sum_{i=0}^{H-1} x^T(t + i + 1) Q x(t + i + 1) + R u^2(t + i).$$

It is assumed that the plant input is scalar, and hence R is a positive scalar weight.

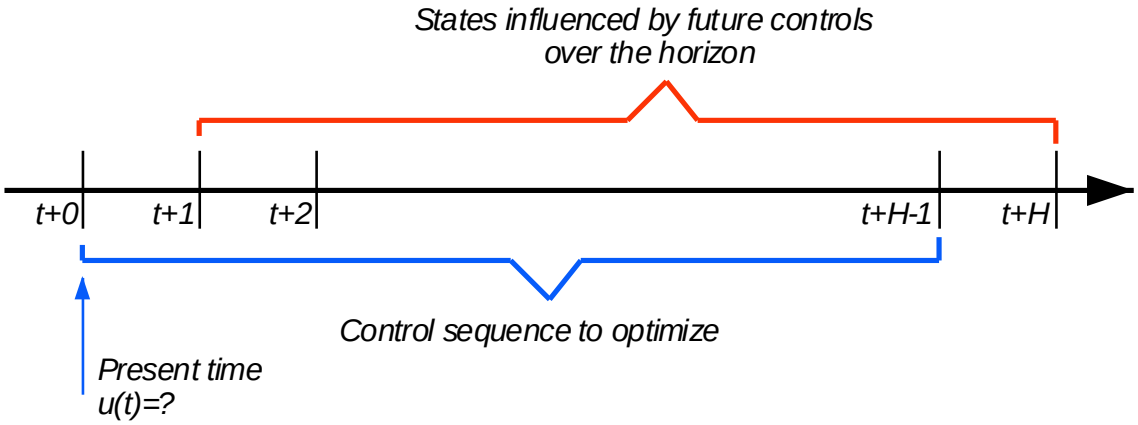

Figure P2-1. The control sequence to optimize in the RH cost and the corresponding sequence of states that is influenced by it.

By selecting $Q = C^T C$, where C is a vector in the output equation

$$y(t) = Cx(t),$$

the cost function penalizes the plant output and becomes

$$J_{RH}(u;t) = \sum_{i=0}^{H-1} y^2(t+i+1) + Ru^2(t+i).$$

Figure P2-1 shows the time synchronization of the variables in this cost function. "We" are at the present time t and want to compute the control sample $u(t)$ to apply to the plant. For that sake, we consider the outputs (or the states) between $t+1$ and $t+H$. Since the plant has a unit delay between applying a control sample and its influence on the state and the output, this set of future outputs is influenced by the control inputs between t and $t+H-1$.

The outcome of the optimization is, therefore, the sequence of future control inputs between t and $t+H-1$. According to the receding horizon strategy, of this whole control sequence, only the first element, $u(t)$ is applied to the plant, the whole procedure being repeated at time $t+1$ (where the present time will be $t+1$, the control sequence with respect to which the optimization is performed is $u(t+1), \dots, u(t+H)$, and the output sequence will be $y(t+2), \dots, u(t+H+1)$) and so on.

In order to simplify the notation, it is assumed, without loss of generality, that $t = 0$.

Define the vectors

$$Y = \begin{bmatrix} y(1) \\ \vdots \\ y(H) \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} u(0) \\ \vdots \\ u(H-1) \end{bmatrix}.$$

With these definitions, the receding horizon cost can be written as

$$J_{RH} = Y^T Y + R U^T U.$$

Using the state model, the vector of future output samples, Y, can be expressed as a function of the vector of future control samples, U, and of the initial state, thereby resulting in a function of U. In order to follow this procedure, iterate the state model to compute future values of the state, yielding (remember that time 0 actually corresponds to present time t; b is used instead of B to emphasize that u is scalar) the following predictive models for the state

$$x(1) = Ax(0) + Bu(0),$$

$$x(2) = A^2 x(0) + Abu(0) + bu(1),$$

$$x(3) = A^3 x(0) + A^2 bu(0) + Abu(1) + bu(2),$$

or, in general,

$$x(i) = A^i x(0) + A^{i-1} bu(0) + A^{i-2} bu(1) + \cdots + bu(i-1).$$

Predictive models for the output are obtained by multiplying by C, yielding

$$y(i) = CA^i x(0) + CA^{i-1} bu(0) + CA^{i-2} bu(1) + \cdots + Cbu(i-1).$$

This set of predictive models, for $i = 1, \cdots, H$, can be written in matrix form as

$$\begin{bmatrix} y(1) \\ y(2) \\ y(3) \\ \vdots \\ y(H) \end{bmatrix} = \begin{bmatrix} Cb & 0 & 0 & \cdots & 0 \\ CAb & Cb & 0 & \cdots & 0 \\ CA^2 b & CAb & Cb & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ CA^{H-1} b & CA^{H-2} b & \cdots & CAb & Cb \end{bmatrix} \begin{bmatrix} u(0) \\ u(1) \\ u(2) \\ \vdots \\ u(H-1) \end{bmatrix} + \begin{bmatrix} CA \\ CA^2 \\ CA^3 \\ \vdots \\ CA^H \end{bmatrix} x(0).$$

Define the matrices

$$W := \begin{bmatrix} Cb & 0 & 0 & \cdots & 0 \\ CAb & Cb & 0 & \cdots & 0 \\ CA^2b & CAb & Cb & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ CA^{H-1}b & CA^{H-2}b & \cdots & CAb & Cb \end{bmatrix}, \qquad \Pi := \begin{bmatrix} CA \\ CA^2 \\ CA^3 \\ \vdots \\ CA^H \end{bmatrix}.$$

With these definitions, the pencil of output predictive models can be written as

$$Y = WU + \Pi \mathrm{x}(0).$$

Inserting these predictive models in the cost

$$J_{RH} = Y^T Y + RU^T U,$$

this same cost is written as

$$J_{RH} = (U^T W^T + x^T(0)\Pi^T)(WU + \Pi x(0)) + RU^T U$$

or

$$J_{RH} = U^T(W^T W + RI)U + 2x^T(0)\Pi^T WU + x^T(0)\Pi^T \Pi x(0),$$

which is a quadratic form in U. To simplify the notation, define the matrix

$$M := W^T W + RI.$$

With this definition, the finite horizon cost is written

$$J_{RH} = U^T MU + 2x^T(0)\Pi^T WU + x^T(0)\Pi^T \Pi x(0).$$

The gradient of J_{RH} with respect to U, is[2]

$$\nabla_U J_{RH} = 2U^T M + 2x^T(0)\Pi^T W.$$

Equating the gradient to zero, yields the following equation, satisfied by the optimal control sequence

$$U^T M + x^T(0)\Pi^T W = 0.$$

Transposing, and solving this equation with respect to U, yields the optimal control sequence

[2] The gradient with respect to x of $x^T Ax$ is $2x^T A$; the gradient with respect to x of $b^T x$ is b^T.

$$U^* = -M^{-1}W^T\Pi x(0).$$

According to the receding horizon strategy, only the first entry of this sequence is applied to the plant. Recalling that $x(0)$, actually, corresponds to $x(t)$, and that $u(0)$ corresponds to $u(t)$, the optimal receding horizon control is given by the state feedback

$$u(t) = -K_{RH}x(t),$$

with the optimal feedback gain given by

$$K_{RH} = e_1 M^{-1}W^T\Pi,$$

where

$$e_1 = [1 \quad 0 \quad \cdots \quad 0]$$

is a row vector of dimension H.

💻 Work to perform

Consider the open-loop unstable 1st order plant

$$x(t+1) = 1{,}2\, x(t) + u(t).$$

For this plant, develop a Matlab script to answer the following questions:

1) Compute the optimal LQ state feedback gain;
2) Compute the optimal receding horizon gain for different values of the horizon H. Make a plot of the gain as a function of H and superimpose it on a line that corresponds to the optimal LQ gain.
3) Make a plot of the absolute value of the closed-loop eigenvalue (the eigenvalue of $A - bK$) and compare it with the stability boundary.
4) Discuss the advantage of enlarging the horizon H in relation to the above example. Compare the RH and LQ gains.
5) Repeat the study for the open-loop stable 1st order plant

$$x(t+1) = 0{,}8\, x(t) + u(t).$$

In relation to stability, for which plant is it more advantageous to use enlarged values of the horizon H?

In the above study, take $Q = 1$ and select values of R that you consider that yield representative results. Try with small and large values of R (you must decide the actual values. The maximum range of H must also be decided by you.

P3 – Basics on predictive control

Motivation

Previous activities, P1 and P2, illustrate different components and concepts associated to predictive control (MPC). Although one might develop software from scratch to implement adaptive control algorithms (for instance to implement advanced features or to optimize the code), it is convenient to use a software package that embeds MPC algorithms and that can be configured for a particular application by defining the plant model and the cost within certain classes (such as the matrices A, B, and C in a linear model, or the quadratic cost parameters, like the weight matrices Q and R, the horizon H, etc.). Although they reduce the possibilities that can be tried, these software packages have the advantage of allowing fast testing and prototyping, evaluating different possibilities before the final configuration of the algorithm is decided and implemented in a (perhaps dedicated) computer for their production run.

There are available several MPC software packages, some freely available, and some proprietary.

Objectives

The objective of this activity is to get familiar to the configuration and use of the MPC software package MPCtools 1.0. This package has been developed by the University of Lund for educational purposes and is free to be use. This package assumes linear plant dynamics and quadratic costs. See the reference below for the original documents and site.

Installing the MPCtools package

The MPCtools 1.0 package is available in Fénix or at the original site from Lund University

https://www.control.lth.se/research/tools-and-software/mpctools/

This software is supported by a Reference Manual (also available in Fénix) and consists of a set of Matlab scripts. Since MPCtools 1.0 is freeware, it can be freely installed in the student's personal computer, without the need to pay any licence or to follow a registration procedure. For that sake, simply copy the MPCtools scripts to a folder. When referring to an MPCtools function, one must indicate the path to the folder. If the project being developed is in a child folder of the folder containing MPCtools scripts, simply add in the beginning of your program

```
path(path,'../')
```

This instruction includes in the Matlab path the path to the directory that contain MPCtools scripts.

Further information can be seen in section 2, pages 5, 6 of the MPCtools Reference Manual [1].

A useful example

This example shows how to use the software of MPVtools 1.0 to configure and simulate MPC control of a plant.

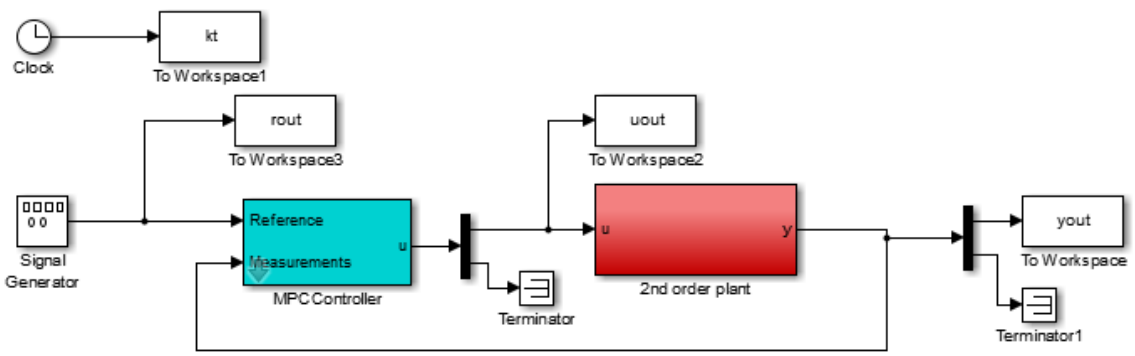

Figure P3-1. SIMULINK block diagram for the MPC control of a 2nd order plant.

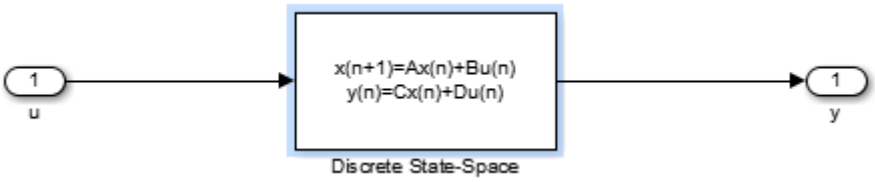

Figure P3-2. SIMULINK block diagram that describes the plant as a subsystem (detail of the red coloured block in figure P3-1).

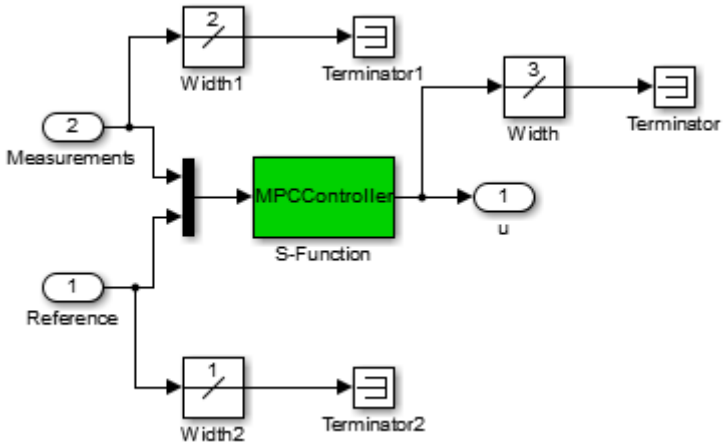

Figure P3-3. SIMULINK block diagram that describes the MPC controller (detail of the cyan coloured block in figure P3-1).

The simulation is done with the SIMULINK block diagram shown in figure P3-1 (file *Oscillating.mdl*). The structure comprises two main blocks, interconnected according to a feedback structure.

The plant is modelled by the red block, labelled 2nd order plant. This block corresponds to a subsystem in which the plant is described as a block diagram and can be described by double- clicking on it, to obtain the diagram in figure P3-2. The input is u (control variable that enters the plant), and the output is y (plant output, that corresponds to the signals measured by the sensors). In this case, the plant is simply described by a SIMULINK state model, shown in figure P3-2. It could be more complex, including nonlinear blocks.

The MPC controller is described in the cyan-coloured block labelled *MPC controller*. By clicking on the lower-left corner, the structure of this block appears, as shown in figure P3-3. Essentially, it consists of an S-function block that, from the values of the

reference and of the plant output computes the plant manipulated variable. The S-function block runs a function called *MPCcontroller.m* that is part of the MPCtools 1.0 package. You don't need to know anything about SIMULINK S-function blocks to use this software. All you need to know is the set of parameters to configure it.

The MPC controller parameters are defined as the input to an initialization function called *MPCInit.m* . This function inputs the parameters that define the MPC parameters (such as the matrices of the linear plant model, cost weights, horizons, and optimization algorithm used; this list will be detailed below) and computes several matrices that are stored in a data object named *md* which will then be used by the function *MPCcontroller.m* that runs the MPC algorithm during the simulation.

Therefore, the structure of the Matlab script to run the simulation comprises the following blocks:

1) Define the linear plant model.
2) Define the MPC parameters.
3) Run *MPCInit.m* to initialize the MPC algorithm (create the data object named *md*).
4) Run the SIMULINK block diagram, that uses the function *MPCcontroller.m*
5) Plot the results.

An example is provided by the Matlab script *MPCoscillating.m* .

The parameters of *MPCInit.m*

The parameters of the Matlab function *MPCInit.m* are based on the parametrization of the MPC problem that is described hereafter.

Plant model: discrete time model with sampling interval h and defined by the state equations

$$x(k+1) = Ax(k) + Bu(k)$$

$$y(k) = C_y x(k)$$

$$z(k) = C_z x(k) + D_z u(k)$$

$$z_c(k) = C_c x(k) + D_c u(k)$$

where,

- $x(k) \in \mathbb{R}^n$ is the state vector at discrete time k.
- $y(k) \in \mathbb{R}^{p_y}$ is the measured output.
- $z(k) \in \mathbb{R}^{p_z}$ is the controlled output.
- $z_c(k) \in \mathbb{R}^{p_c}$ is the constrained output.
- $u(k) \in \mathbb{R}^m$ is the manipulated input.

It is remarked that the package allows to distinguish between the measured output and the controlled output. In the simple cases to be considered hereafter, they are both equal, meaning that $C_z = C_y$ and D_c is a matrix of zeroes of convenient dimension.

The **cost function** is quadratic, written as

$$J(k) = \sum_{i=H_w}^{H_p+H_w-1} \|\hat{z}(k+i|k) - r(k+i|k)\|_Q^2 + \sum_{i=0}^{H_u-1} \|\Delta\hat{u}(k+i|k)\|_R^2.$$

Here, $\|v\|_M^2 \coloneqq v^T M v$ for a matrix M and a vector v.

The index H_w is used to shift the time in which the future variables are accounted along the horizon. Although this index might be useful for plants with pure delay in the control action, here we just consider $H_w = 1$.

The index H_p is the prediction horizon. With the choice $H_w = 1$, its minimum value is 2 (otherwise there is an error message yielded by the software).

The index H_u is the, so called, control horizon, that corresponds to the number of free samples of predicted control action. It is suggested to start with $H_u = H_p$, that is the maximum for this parameter. Reducing H_u, reduces the computational load, but the properties of the controller change.

With the choice $H_w = 1$ and $H_u = H_p$, the cost becomes

$$J(k) = \sum_{i=1}^{H_p} \|\hat{z}(k+i|k) - r(k+i|k)\|_Q^2 + \sum_{i=0}^{H_p-1} \|\Delta\hat{u}(k+i|k)\|_R^2.$$

The matrices $Q = Q^T \succcurlyeq 0$ and $R = R^T \succ 0$ are weighting matrices that are constant the prediction horizon.

In the cost function, the time increments $\Delta u(k) \coloneqq u(k) - u(k-1)$ are penalized, instead of $u(k)$. This allows to track non-zero references without error.

The **constraints** are defined by

$$\Delta u_{min} \leq \Delta u(k) \leq \Delta u_{max}, \quad k \in I_u$$

$$u_{min} \leq u(k) \leq u_{max}, \quad k \in I_u$$

$$z_{min} \leq z_c(k) \leq z_{max}, \quad k \in I_p$$

The sets I_u and I_p contain the samples over the prediction horizon for which the constraints are enforced. These sets are defined by the *MPCinit* function arguments *ublk* and *zblk*. The value suggested are 1 in each case, that corresponds to always considering the constraints active, as well as to include in the cost all the instants along the prediction horizon.

The syntax to call *MPCInit* is

Md=MPCInit(Ad, Bd, Cyd, Czd, Dzd, Ccd, Dcd, Hp, Hw, zblk, Hu, ublk, du_max, du_min, u_max, u_min, z_max, z_min, Q, R, W, V, h, cmode, solver)

Table P3-1 describes the arguments of *MPCinit.*

***MPCinit* arguments**	**Description**
Ad, Bd, Cyd	Plant state model matrices: A_d, B_d, and C_d. The dimensions of these matrices implicitly define the state dimension.
Czd, Dzd	Matrices that define the controlled outputs: C_z and D_z.
Ccd, Dcd	Matrices that define the constrained outputs: C_c and D_c.
Hw, Hp, Hu	Integers that define the prediction and control horizons: H_w, H_p, and H_u. Suggested: $H_w = 1$, $H_p \geq 2$, $H_u = H_p$.
zblk, ublk	Blocking factors that define the sets I_p and I_u. Suggested: zblk=ublk=1.
u_min, u_max	Minimum and maximum values of the control variable, u_{min} and u_{max}.
du_min, du_max	Minimum and maximum values of the increments of the control variable, Δu_{min} and Δu_{max}.
z_min, z_max	Minimum and maximum values of the controlled variable, z_{min} and z_{max}.
Q, R	Weighting matrices for the cost function.
W, V	Noise variances for Kalman filter design, if applicable. Make W=[ ] and V=[ ] if the Kalman filter is not used.
h	Sampling interval (units of the model)
cmode	Integer that specifies the controller mode. If cmode=0, W and V are not used and may set W=[ ] and V=[ ]. The matrix C_y is assumed to be the identity matrix.
solver	String that specifies the optimization solver used for the quadratic programming problem. Possibilities that do not require the Matlab Optimization Toolbox to be installed are qp_as and qp_ip. Use, for instance, solver='qp_as'.

Table P3-1. The arguments of MPCInit.

💻 Work to perform

0) Run the example previously described (script *MPCoscillating.m* that uses the SIMULINK block diagram *Oscillating.mdl*). Change the parameters that define the MPC as well as the constraints. Get familiar with the software structure and the parameters that allow you to configure the algorithm. The script has Hp=3 (that corresponds to a prediction horizon with 2 points). What happens if Hp=2? And if you enlarge Hp? What happens if the constraints on u are tightened? You don't need to write anything on your report on this task number 0. Just use this example as a prototype that you can modify to control other plants and solve P4.

1) Consider the first order open-loop unstable plant

$$x(k+1) = 1.2x(k) + u(k), \qquad y(k) = x(k).$$

Simulate its control with MPC. Illustrate the effect of constraints on the input, on the input rate of change and on the state, as well as the effects of the choice of the cost weights and the horizon.

2) Consider a situation in which the weight R on the control variable is large and there are moderate constraints on the control amplitude. Can you find a situation that shows that enlarging the prediction horizon results in an increase of performance?

P4 – MPC control design for the inverted pendulum

Motivation

MPC is a general control design set of algorithms that can be applied to a variety of plants. Its unique feature is that it allows the explicit inclusion on design of constraints. Since MPC requires the on-line solution of a sequence of optimization problems, it is, in general, computationally heavy and, in the beginning, was conceived for applications in slow processes, in sites where a powerful computer is available, like process control. The first algorithm (DMC, acronym of Dynamic Matrix Control) was invented for the control of distillation columns to process crude oil; the motivation was to obtain a control algorithm that could be easily re-tunned so as to meet different crude characteristics, that was being bought from changing producers, according to market prices, and such that it allows the explicit incorporation of constraints in the control and state variables.

With major progress in algorithms (in control, optimization, and numerical methods), and enormous advances in computing hardware and software, the MPC range of applications has been progressively enlarged to faster plants, including robotics and motor car engine control, for which there are several real time examples described in the literature. Even the increasingly expanding space industry is considering the use of MPC for the control of spacecraft manoeuvres, due to its capacity of embedding safety constraints [2]. Therefore, there is a big motivation to study the design of MPC. The example considered here (equilibration of the inverted pendulum) is motivated to be in tight relation with the attitude control of a landing reusable rocket.

Objectives

The objective of this activity consists of designing an MPC controller to equilibrate an inverted pendulum. Opposite to the previous activities, that consisted of the answer to precise questions, or of training with an MPC package in well defined exercises, in this task the student must take the lead in deciding the design options (within linear plant models and quadratic costs) and to demonstrate the resulting performance and limitations.

The control problem

The plant to control consists of an inverted pendulum, acted by a torque, as shown in figure P4-1. This plant is a simplification of the dynamics of the attitude of a landing rocket, with the angle of the gimbal used to create a torque used as manipulated variable.

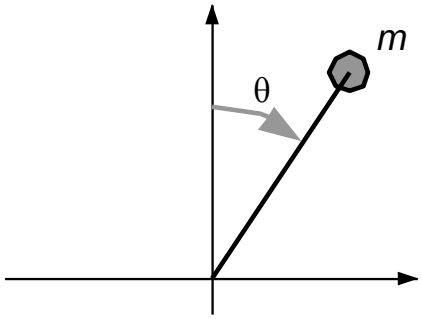

Figure P4-1. The plant to control in P4.

Nonlinear plant model

In relation to figure P4-1, the state of the nonlinear plant is defined as the angular position and velocity of the tip of the pendulum, or

$$x_1 = \theta,$$

$$x_2 = \dot{\theta}.$$

These state variables satisfy the following nonlinear model

$$\dot{x}_1 = x_2,$$

$$\dot{x}_2 = a \sin x_1 - b x_2 + cu,$$

where,

- $a = g/L$ with $g = 9{,}8$ ms^{-2} (gravity acceleration) and $L = 0{,}3$ m (pendulum length);
- $b = k/m$ with $k = 0{,}01$ (compatible units), and $m = 0{,}3$ kg (pendulum mass);
- $c = 1/(mL^2)$;

- The angles are measured in rad;
- u is the torque, measured in Jm.

Linearized plant model

Taken as equilibrium point $\theta = 0$, the state of the nonlinear model is equal to the state of the linearized model. For small deviations with respect to the equilibrium, the linearized model equations are

$$\dot{x}_1 = x_2,$$

$$\dot{x}_2 = ax_1 - bx_2 + cu$$

This model is defined in continuous time. To obtain a model in discrete time, one may use the Matlab function *c2d* (see further details below).

Control objectives

1. Starting from a non-zero initial condition, act on the torque such as to drive the angle to zero.
2. Act on the torque such that the angle tracks a reference of small amplitude.

A base-line example: LQ control

The files *mLinPendLQR.m* and LinPendLQ.slx provide an example of the control of the pendulum using LQ control. Running these files requires, in addition to Matlab and SIMULINK, the control systems toolbox. If you don't have access to this toolbox, it might still be instructive to look at them and remark a few points.

Figure P4-2 shows the SIMULINK block diagram used to simulate the controlled pendulum. A major point to remark is that the plant to control must be defined in continuous time and is described by the nonlinear model. On the other way, the controller is designed based on the linearized model, defined in discrete-time.

To make the interface between the plant (continuous time) and the controller (discrete-time), one must insert Zero-Order Hold blocks in the state variables (see figure P4-2).

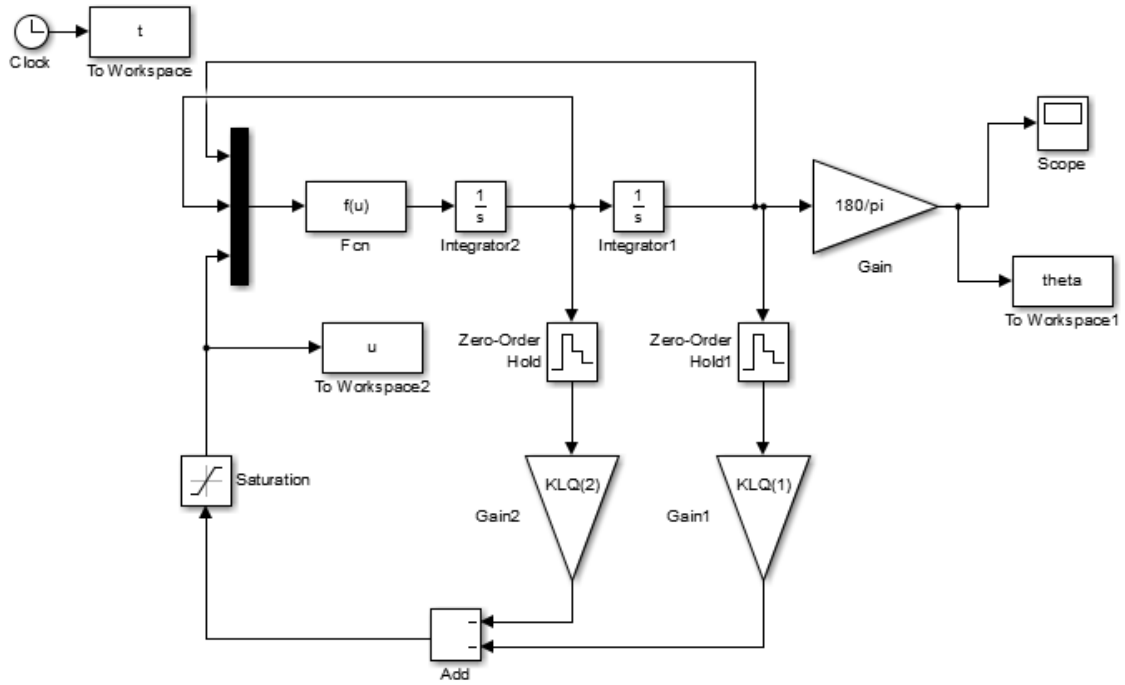

Figure P4-2. Controlling the pendulum with LQ control.

Work to perform

1. By adapting the examples in P3 (see in particular figure P3-1), write the software required to simulate the control of the nonlinear inverted pendulum using MPTtools 1.0. Don't forget that your controller is based on a linearized, discrete time model, but the simulation tests are made with a nonlinear, continuous time, model (the red block in figure P3-1).
2. Discuss the configuration of the MPC parameters. It is suggested that you start by ignoring the saturations and take Q as the identity matrix and R small. The script on LQ control suggests a value for the sampling interval. Don't forget to program the sampling interval that you are using in the ZOH bocks of the SIMULINK diagram. After you have made your sw package to run in good order, you may then include constraints and consider more complicated situations. Of course time plots are quite important in documenting your results, but plots that provide a global view of the controlled system and its tuning are much more valuable, for instance plots that show the effect on a merit figure of selecting R or the horizon.

3. Discuss the limitations imposed by hard-limiting the control action. Compare the effect of hard constraints when MPC and LQ control is being used. Discuss the effect of enlarging the horizon. Make intuitive interpretations. Consider a situation in which the pendulum starts with some inclination but, with the control constraint considered, it is such that the controller can equilibrate it. What happens when the initial angle deviates more with respect to the vertical? You might want to do a discussion (based on simulations) of the basin of attraction of the vertical position with respect to the initial conditions.
4. Discuss the effect of disturbances. A disturbance of special relevance simulates a wind gust, in which the wind pressure evolves, in a deterministic way, according to figure P4-3.

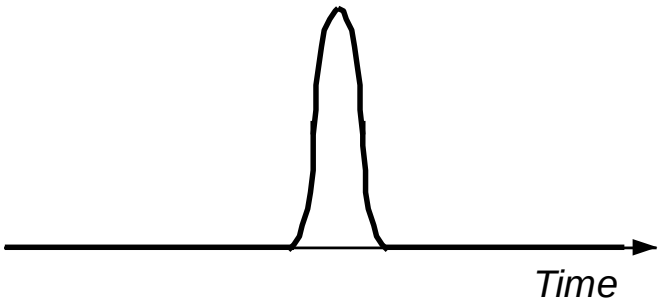

Figure P4-3. Time evolution of the pressure created by a wind-gust.

References

[1] Johan Åkesson. MPCtools 1.0 – Reference Manual. Department of Automatic Control, Lund Institute of Technology, 2006. Available in Fénix or in https://www.control.lth.se/research/tools-and-software/mpctools/

[2] A. Botelho, B. Parreira, P. N. Rosa and J. M. Lemos. *Predictive Control for Spacecraft Rendezvous.* Springer, 2021.

Appendix 1 – Good practice rules to write the report

1. **Adequacy of the answers** to the questions in the work guide
   a) The answers must fully respond to the intended
   b) Conciseness. Use concise text, but that fully expresses the answer
   c) Clarity. The text and its articulation with the figures included in the report must be clear. If variables not defined in the statement are used, they must be defined in the report.
   d) Technical correction. There should be neither technical errors in the answers, nor errors in the use of concepts.
   e) Critical analysis of results. Do the results make sense given what is known and/or when compared to each other? What is the interpretation of the results?
2. **Imagination and originality** in the answers and in the way they are presented.
3. **Correction of the text** and elegance of the style.
   a) There should be no spelling or syntax errors.
   b) Preferably use short and clear sentences
   c) Well articulated text.
   d) Proper punctuation, including correct punctuation of equations.

4. **Readable and informative graphics and plots**
   a) Identify the variables that correspond to each axis and, if appropriate, the units in which they are expressed.
   b) Proper axes font (not too small or too large; type 14 is suggested).
   c) Use appropriate scales, in order to show the important part of the evolution of the curves and/or the global evolution of the variables, depending on the situation.
   d) Do not hide important parts of curves with labels.
   e) Do not use a range of variables that is equal to the range of data. In particular, if the variable to plot has periods during which it is constant, do not select the range of the axis to coincide with this constant. For instance,

when plotting a square wave between 0 and 1, do note use vertical axes between 0 and 1, but include a small margin, say, axis between -0.1 and 1.1.

f) Size of the graphics in the document suitable for what you want to show.

g) Concise graph legends that elucidate the main aspect of the graph.

5. Clear, well-structured and documented programs

1. Correction of programs.
2. Good structure of programs.
3. Adequate and well distributed comments throughout the program.
4. Good program intelligibility.
5. Adequate and short names of variables, explanation of variables with comments.
6. Avoid using i and j as variables in your program because they are preassigned to the unit imaginary number (type i in the Matlab command window and see the answer; repeat for j). You can use ii and jj as counters, for instance.

– End of document –