Position estimation and control of an autonomous vehicle

Joana Maria Comas Martí
EPFL[1]-ETSEIB[2]

Supervisor: MER Denis Gillet
Assistants: Levente Bodizs and Davide Buccieri

2005

[1]École Polytechnique Fédérale de Lausanne
[2]Escola Tècnica Superior d'Enginyeria Industrial de Barcelona

This master thesis is presented to the Laboratoire d'Automatique (LA) of the École Polytechnique Fédérale de Lausanne (EPFL) by Joana Maria Comas Martí

Joana Maria Comas Martí

Lausanne, Switzerland; 30/06/2005

Acknowledgements

First I would like to thank Dr. Denis Gillet who gave me the opportunity to work in the Laboratoire d'Automatique (LA) of the École Polytechnique Fédérale de Lausanne (EPFL).

I greatly appreciate the time, help and support my assistants Levente Bodizs and Davide Buccieri offered to me. I also want to thank the help with LabVIEW from Christoph Salzmann and the work Philippe Cuanillon did in the experimental setup.

Finally I would like to say that I really appreciate the support both academically and personally from all the members of the LA, specially from Sébastien Thiery and Mark Butcher.

Contents

List of Figures

List of Tables

1 Introduction

This master thesis has been developed at the Laboratoire d'Automatique (LA) of the École Polytechnique Fédérale de Lausanne (EPFL).

The aim of this project is to improve the tracking performance of a nonholonomic vehicle. With this purpose, two different tasks are done:

1. Position estimation: a positioning system based on an external camera gets 3 on-line measurements: (x, y coordinates and θ orientation). Then, to improve the accuracy of these measurements, a Kalman filter is used.

2. Control: two control structures are tested.

   (a) Classical Model Predictive Control (MPC)

   (b) Estimator Based Control Using Flatness (EBCUF)

Both parts of the project have been developed following these steps:

- Simulation in Matlab
- Simulation in LabVIEW
- Implementation using LabVIEW to give the command to a small mobile robot, the Khepera, shown in figure 1.

Finally, both strategies are compared, giving the advantages and disadvantages in terms of performance and implementability.

The reason to use a Khepera and control it with LabVIEW is the ease it presents to study different control algorithms and to compare their performances in real-time applications. Moreover some previous projects have already been developed using this robot, which is something that assures a certain expertise of LA in this domain.

There are three master projects that have already been written at the Institut d'Automatique of the EPFL by [Herrera, 1999], [Seba, 2000] and [Niederberger, 2002]. The common part of all of them is that they implemented a Model Predictive Control algorithm for the Khepera robot.

This project will use image treatment algorithms based on the ideas of [Herrera, 1999] to get the on-line measurements from the camera and one of the Model Predictive Control algorithms proposed by [Niederberger, 2002].

Concerning the Estimator Based Control Using Flatness, the theoretical part and Matlab simulations have been developed by Davide Bucceri, PhD student at LA. However, the implementation has not been previously done and it is the contribution of this project.

Although there were codes which already existed for the different parts of the project, all those that this thesis uses have been re-written, both the ones for Matlab and the ones for LabVIEW.

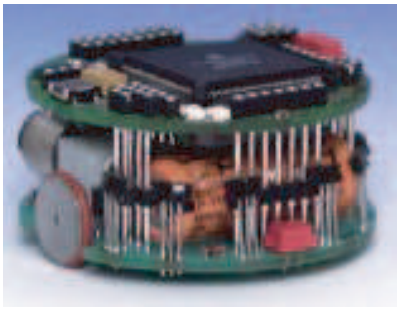

Figure 1: Image of the Khepera robot.

2 Modeling the system

2.1 Continuous time model

Physical systems can be represented by different models. These models can be classified into *kinematic* and *dynamic* models. The first ones only consider the movement laws applied to the system, while the second ones take also into account the masses and forces appearing in the system, i. e., they also consider Newton's law $F = ma$. When doing the assumption of having a negligible mass, it is implicit in it that only the kinematics will be considered, but not the dynamics.

Another criteria to classify the models is the way they deal with the time. In this case, models can be either *continuous time* or *discrete time*.

Considering a knife edge **system**, with negligible mass, moving on a planar surface we have the next kinematic differential equations (continuous time model):

$$\dot{x}(t) = v(t)cos\theta(t) \tag{1a}$$
$$\dot{y}(t) = v(t)sin\theta(t) \tag{1b}$$
$$\dot{\theta}(t) = \omega(t) \tag{1c}$$

where $v(t)$ is the linear speed and $\omega(t)$ is the angular speed of the knife edge.

In matrix form, the system is represented by

$$\dot{\mathbf{x}}(t) = \mathbf{x}(t) + \mathbf{B}(\theta(t))\mathbf{u}(t) \tag{2a}$$

where

$$\mathbf{x}(t) = \begin{pmatrix} x(t) \\ y(t) \\ \theta(t) \end{pmatrix} \tag{2b}$$
$$\mathbf{B}(\theta(t)) = \begin{pmatrix} cos\theta(t) & 0 \\ sin\theta(t) & 0 \\ 0 & 1 \end{pmatrix} \tag{2c}$$
$$\mathbf{u}(t) = \begin{pmatrix} v(t) \\ \omega(t) \end{pmatrix} \tag{2d}$$

Note: The orientation θ is given by the angle formed by the x axis and the direction of the linear speed. It is not an angle measured taking the origin or any fix point as reference.

Something interesting to note is the fact that the first two components of the state $x(t)$ and $y(t)$ depend only on the input $v(t)$, while the third one, $\theta(t)$ depends only on $\omega(t)$. However, this does **not** mean that the model is **decoupled** as [Herrera, 1999] and [Niederberger, 2002] mentioned. Having a decoupled model means that a part of the state depends only on some components of the input; and the rest of the state

depends on the other components of the input. Therefore the controls of each part of the state are completely independent.

In this case, however, the control of $x(t)$ and $y(t)$ is related to $\theta(t)$. Only when the linear speed is zero, the system could be considered decoupled.

2.2 Properties of the system

Assumption: this continuous time model is assumed to be an accurate representation of the system. Therefore, the properties of the model, are also assumed to be valid for the system.

The system is **non-linear**, **time invariant** and **causal**. The reasons are the following ones:

Given a system in the form:

$$y = f(x) \tag{3}$$

it is said to be linear if it verifies the superposition principle:

$$\underbrace{f(\alpha x_1 + \beta x_2)}_{\mathbf{(a)}} = \underbrace{\alpha f(x_1) + \beta f(x_2)}_{\mathbf{(b)}} \tag{4}$$

For our system, two states would be

$$\mathbf{x_1} = \begin{pmatrix} x_1 \\ y_1 \\ \theta_1 \end{pmatrix}$$

and

$$\mathbf{x_2} = \begin{pmatrix} x_2 \\ y_2 \\ \theta_2 \end{pmatrix}$$

Then, if we check the differential equation corresponding to the first component of the state, we get that

$$(\mathbf{a}) = (x_1 \dot{+} x_2) = vcos(\theta_1 + \theta_2) \tag{5a}$$

$$(\mathbf{b}) = \dot{x}_1 + \dot{x}_2 = v(cos(\theta_1) + cos(\theta_2)) \tag{5b}$$

Since $cos(\theta_1 + \theta_2)$ is not necessarily equal to $cos(\theta_1) + cos(\theta_2)$, the system is not linear, so it is **non-linear**.

A general form to represent the system would be

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{6}$$

If this function $\mathbf{f}(\mathbf{x}, \mathbf{u})$ is constant in time, which is the case of (2a), the system is said to be **time invariant**.

Finally, the model is **causal** because the derivatives of the state depend only on the actual state and the actual input.

Now, when it comes to get a **discrete time model**, approximations need to be done. An option is to suppose that the trajectory followed by the system between two consecutive instants is either linear (section 2.3) or circular (section 2.4). These two options were considered by [Herrera, 1999], [Seba, 2000] and [Niederberger, 2002]. This project restates the models used by them, correcting some details, adding explicitly the approximations done to get these models and trying to show the models in the simplest way, while keeping completeness.

2.3 Model based on segments

Getting a kinematic model in discrete time based on straight segments means assuming the system is following a straight line between two consecutive sampling instants k and $k+1$. To be more precise, what is assumed

1. to get $x(k+1)$ and $y(k+1)$ is:
   (a) linear speed $v = v(k)$ (constant during a sampling period)
   (b) orientation of the system $\theta = \theta(k)$ (constant during a sampling period)
2. to get $\theta(k+1)$ is: the angular speed $\omega = \omega(k)$ (constant during a sampling period)

Then, to get this discrete time model, the equations of the system (1) are taken as starting point and the Euler forward-difference approximation is applied to them to obtain the following model

$$x(k+1) = x(k) + hv(k)\cos\theta(k) \tag{7a}$$
$$y(k+1) = y(k) + hv(k)\sin\theta(k) \tag{7b}$$
$$\theta(k+1) = \theta(k) + h\omega(k) \tag{7c}$$

where h is the sampling period.
Or in matrix form

$$\mathbf{x}(k+1) = \mathbf{x}(k) + \mathbf{B}(\theta(k))\mathbf{u}(k) \tag{8a}$$

where $\mathbf{x}(k)$, $\mathbf{B}(\theta(k))$ and $\mathbf{v}(k)$, are analogous to (2) but discrete time variables and not continuous time ones.

$$\mathbf{x}(k) = \begin{pmatrix} x(k) \\ y(k) \\ \theta(k) \end{pmatrix} \tag{8b}$$
$$\mathbf{B}(\theta(k)) = \begin{pmatrix} hcos\theta(k) & 0 \\ hsin\theta(k) & 0 \\ 0 & h \end{pmatrix} \tag{8c}$$
$$\mathbf{u}(k) = \begin{pmatrix} v(k) \\ \omega(k) \end{pmatrix} \tag{8d}$$

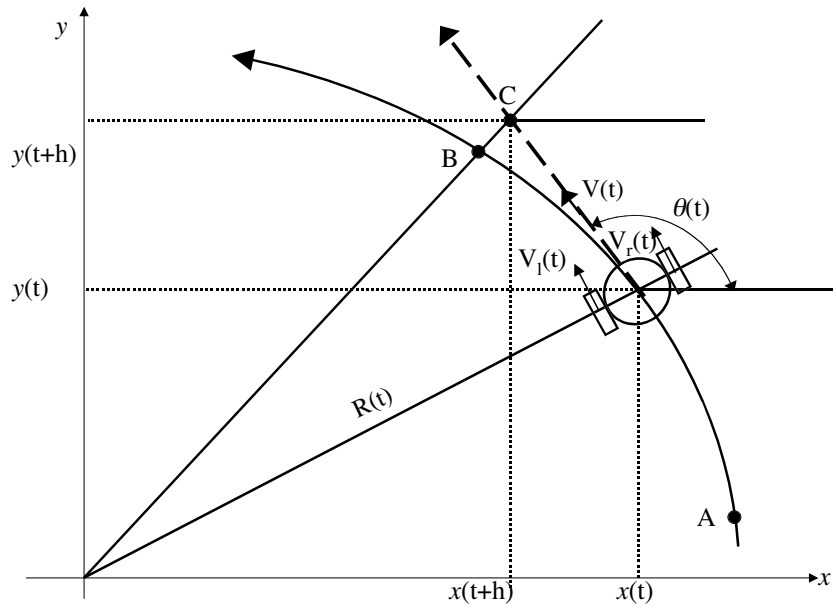

Figure 2: Scheme of the model based on linear segments. Real trajectory: AB; trajectory given by the model: AC; error of the model in a time h: BC. **Note:** The center of the curve is not necessarily the origin, as in [Niederberger, 2002].

Also in this discrete time model the first two components of the state $x(k)$ and $y(k)$ depend only on the input $v(k)$, while the third one, $\theta(k)$ depends only on $\omega(k)$.

As mentioned before, v is the linear speed and ω is the angular speed of the knife edge. If we consider a vehicle of two wheels, like the Khepera, the inputs could be for instance the linear speeds of the right and left wheels, V_r and V_l respectively.

For the experimental implementation using the Khepera, these two speeds, V_r and V_l, are actually the inputs needed. However, since the model with v and ω is simpler and more intuitive, we will work with it while it is possible.

The transformation from one input to the other is given by the simple equations that follow:

$$v(k) = \frac{1}{2}(V_r(k) + V_l(k)) \tag{9a}$$

$$\omega(k) = \frac{1}{L}(V_r(k) - V_l(k)) \tag{9b}$$

2.4 Model based on arcs

The second discrete time model is based on the approximation that assumes that the system follows a circular trajectory between two consecutive sampling instants k and $k+1$.

In this case the approximations consist of

1. To get $x(k+1)$ and $y(k+1)$:
   (a) linear speed $v = v(k)$ (constant during a sampling period)
   (b) radius of the curve $R = R(k) = \frac{\omega(k)}{v(k)}$ (constant during a sampling period)

   Hence, for this part, it is equivalent to say that $v = v(k)$ and $\omega = \omega(k)$ are constant
2. To get $\theta(k+1)$: the angular speed $\omega = \omega(k)$ (constant during a sampling period)

As we can observe, one approximation less than for the model of linear segments ($\theta = \theta(k)$) is required.

The model that results when assuming the above is

$$x(k+1) = x(k) + \frac{\omega(k)}{v(k)}(\sin\theta(k+1) - \sin\theta(k)) \tag{10a}$$

$$y(k+1) = y(k) - \frac{\omega(k)}{v(k)}(\cos\theta(k+1) - \cos\theta(k)) \tag{10b}$$

$$\theta(k+1) = \theta(k) + h\omega(k) \tag{10c}$$

where h is the sampling period.

In this case, getting a matrix form, maintaining v and ω as inputs, is not possible due to the division of the inputs. However, an option to get the matrix form would be to use R and ω as inputs. The model would be in the form

$$\begin{pmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{pmatrix} = \begin{pmatrix} x(k) \\ y(k) \\ \theta(k) \end{pmatrix} + \begin{pmatrix} \sin\theta(k+1) - \sin\theta(k) & 0 \\ -\cos\theta(k+1) + \cos\theta(k) & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R(k) \\ \omega(k) \end{pmatrix} \tag{11}$$

Or in a more compact form

$$\mathbf{x}(k+1) = \mathbf{x}(k) + \mathbf{B}(\theta(k))\mathbf{u}(k) \tag{12a}$$

where $\mathbf{x}(k)$, $\mathbf{B}(\theta(k))$ and $\mathbf{v}(k)$ are

$$\mathbf{x}(k) = \begin{pmatrix} x(k) \\ y(k) \\ \theta(k) \end{pmatrix} \tag{12b}$$

$$\mathbf{B}(\theta(k)) = \begin{pmatrix} \sin\theta(k+1) - \sin\theta(k) & 0 \\ -\cos\theta(k+1) + \cos\theta(k) & 0 \\ 0 & 1 \end{pmatrix} \tag{12c}$$

$$\mathbf{u}(k) = \begin{pmatrix} R(k) \\ \omega(k) \end{pmatrix} \tag{12d}$$

Using the $R(k)$ and $\omega(k)$, the first two components of the state $x(k)$ and $y(k)$ depend only on the input $R(k)$, while the third one, $\theta(k)$ depends only on $\omega(k)$.

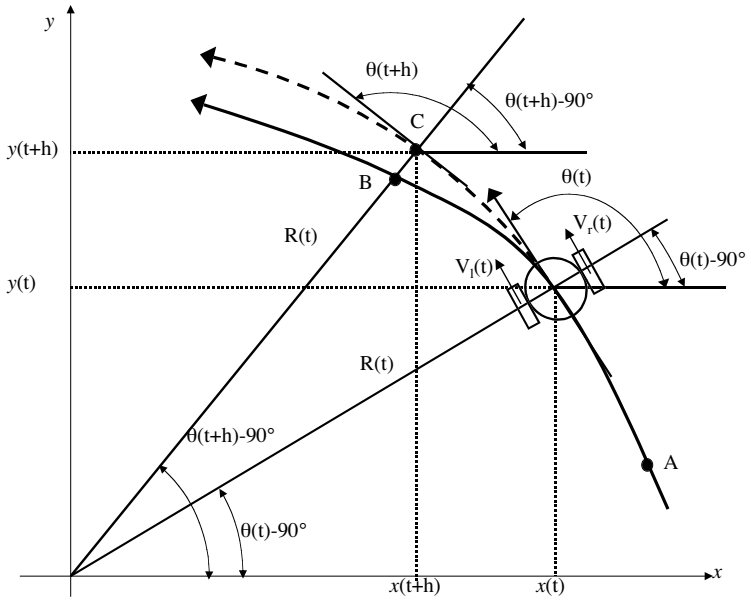

Figure 3: Scheme of the model based on circular arcs. Real trajectory: AB; trajectory given by the model: AC; error of the model in a time h: BC. **Note:** The center of the curve is not necessarily the origin, as in [Niederberger, 2002].

Note: all the models presented in this project do not include the inertia of the system. Some work including it was done by [Niederberger, 2002].

2.5 Model choice

Both the model based on segments (7) and the model based on arcs (10) are:

- Non-linear
- Causal
- Time invariant

Hence, these properties have not contributed to the model choice. What has been taken into account is:

1. Both models have been already used for experimentation by [Herrera, 1999],[Seba, 2000] and [Niederberger, 2002], and the results prove they provide the same level of accuracy.

2. The model based on segments, which is used for the Model Predictive Control, has the same inputs as the continuous model, which is used for the Estimator Based Control Using Flatness.

3. The inputs used by (1) and (7), that is v and ω, are more intuitive than using R and ω.

These are not extremely important reasons. However, previous work [Herrera, 1999], [Seba, 2000] and [Niederberger, 2002], prove that the choice of the model is not that important either, in order to get good results.

3 Model Predictive Control

For the control part of the project, two types of control are used. The first one is a Model Predictive Control (MPC) and the second one, an Estimator Based Control Using Flatness (EBCUF).

First of all, the general theoretical principles that are applied are presented. After that, the reasons to choose the algorithm implemented are presented.

3.1 Introduction

Model (Based) Predictive Control (MPC) is the class of advanced control techniques most widely applied in the process industries. It was developed in the process industries in the 1960's and 70's.

In many control problems, it is desired to design a stabilizing feedback such that a performance criterion is minimized while satisfying constraints on the controls (inputs)and the states. Ideally one would look for a closed solution for the feedback satisfying the constraints while minimizing the performance. However, often the closed solution cannot be found analytically, even in the unconstrained case since it involves the solution of the corresponding Hamilton Jacobi-Bellmann equations. One approach to circumvent this problem is the repeated solution of an open-loop optimal control problem for a given state. The first part of the resulting open-loop input signal is implemented and the whole process is repeated. Control approaches using this strategy are referred to as Model Predictive Control (MPC), moving horizon control or receding horizon control [Allgöwer et al., 2004].

This means that MPC does not refer to a specific control strategy but a very ample range of control methods which make an explicit use of a model of the process to obtain the control signal by minimizing an objective function. This objective or cost function considers the deviations from a desired trajectory.

An advantage of the MPC approach is that to minimize the cost function, constraints can also be taken into account.

Therefore, the basic principle of MPC is to solve an open-loop optimal control problem at each time step. Feedback is handled by updating the model at each time step. Normally the disturbance term is corrected by using the measurements of the output at the current instant.

The ideas common to all the predictive control strategies are [Camacho and Bordons, 1999]:

- Explicit use of a model to predict the process output at future time instants (horizon).
- Calculation of a control sequence minimizing an objective function.
- Receding strategy, so that at each instant the horizon is displaced towards the future, which involves the application of the first control signal of the sequence calculated at each step.

The differences between the various MPC algorithms are [Camacho and Bordons, 1999]:

- Model used to represent the process.
- Noise
- Cost function to be minimized

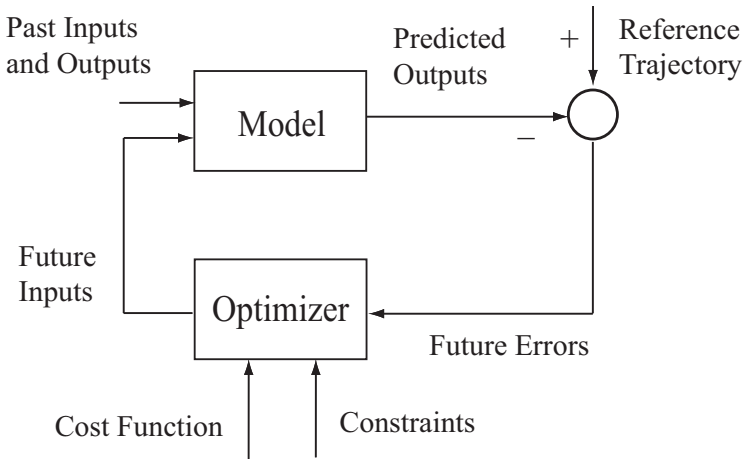

Figure 4: Basic structure of MPC [Camacho and Bordons, 1999]

The following theoretical points are more extensively explained in [Allgöwer et al., 2004].

Linear MPC refers to those MPC schemes which considers linear models to describe the process and linear constraints on states and inputs. Even if the system is linear, the closed loop dynamics are nonlinear due to the presence of constraints.

Nonlinear MPC (NMPC) refers to MPC schemes that are based on nonlinear models and/or consider a non-quadratic cost-function and general nonlinear constraints.

If there were no disturbances and no model-plant mismatch, and if the optimization problem could be solved over an infinite horizon, then the input signal found at the initial instant could be applied open-loop to the system for all the following instants. However, due to disturbances and model-plant mismatch, the actual system behavior is different from the predicted one. To incorporate feedback, the optimal open-loop input is implemented only until the next sampling instant.

The reason to consider finite prediction and control horizon, is the fact that normally, the solution of a nonlinear infinite horizon optimal control problem cannot be calculated (sufficiently fast). In this case, the actual closed-loop input and states will differ from the predicted open-loop ones, even if no model-plant mismatch and no disturbances are present.

The difference of the predicted values and the closed-loop values has two consequences:

1. In general, it is not true that the repeated minimization over a moving finite horizon leads to an optimal solution for the infinite horizon problem. The solutions will differ significantly if a short finite horizon is chosen.

2. There is in general no guarantee that the closed-loop system will be stable, however in many cases, as the one in this project, it is.

3.2 MPC Strategy

In order to explain with some detail the MPC strategy, it might be good to start by presenting a typical cost function to be minimized [Niederberger, 2002]. Once it is presented, the explanation of the way the optimization process is done might be easier.

One of the most common cost functions to be minimized at each time instant is:

$$J_2 = \sum_{i=1}^{N_p} Q_i(r_i - \hat{x}_i)^2 + \sum_{i=1}^{N_c} \Lambda_i(\Delta\hat{u}_i)^2 \tag{13}$$

Which can be written in the more compact vector-matrix form

$$J_2 = (\mathbf{r} - \hat{\mathbf{x}})^{\mathbf{T}}\mathbf{Q}(\mathbf{r} - \hat{\mathbf{x}}) + \mathbf{\Delta\hat{u}}^{\mathbf{T}}\mathbf{\Lambda}\mathbf{\Delta\hat{u}} \tag{14}$$

with

$$\begin{aligned} \mathbf{r}^T &= [r(k+1)^T \;\; r(k+2)^T \;\; \cdots \;\; r(k+N_p)^T] \\ \hat{\mathbf{x}}^T &= [\hat{x}(k+1)^T \;\; \hat{x}(k+2)^T \;\; \cdots \;\; \hat{x}(k+N_p)^T] \\ \mathbf{\Delta\hat{u}}^T &= [\Delta\hat{u}(k)^T \;\; \Delta\hat{u}(k+1)^T \;\; \cdots \;\; \Delta\hat{u}(k+N_c)^T] \end{aligned}$$

where N_p is the prediction horizon and N_c is the control horizon, $\mathbf{r}$ is a vector containing the reference values for times $k+1$ through $k+N_p$, $\hat{\mathbf{x}}$ is a vector containing the estimated values for the states for times $k+1$ through $k+N_p$, and $\mathbf{\Delta\hat{u}}$ is a vector containing the increment in the input values for instants k through $k+N_c$, where $\Delta u(k+j) = u(k+j) - u(k+j-1)$. The weighting matrix Q is a diagonal matrix of the form $Q = diag(q_1, q_2, \ldots, q_{N_p})$, and Λ is a diagonal matrix of the form $\Lambda = diag(\lambda_1, \lambda_2, \ldots, \lambda_{N_c})$. Even though (13) is valid for a single-input/single-output system, the generalization (14) is valid for a multiple-input/multiple-output system.

Note: In equations (13) and (14), the reference values r_i and $\mathbf{r}$, respectively, are the set points for the state. This is what the authors of references [Herrera, 1999], [Niederberger, 2002] and [Allgöwer et al., 2004] consider. On the other hand, there are some authors who work with the same cost function but introduce in it the reference values for the output. It is the case of [Camacho and Bordons, 1999], [Nikolaou, 2001] and [Rawlings, 1999]. They do so, because they are indeed assuming that the state of the plant is perfectly measured, i.e., the state and the output are equivalent. So, all the state information is required for the prediction. Then, the full

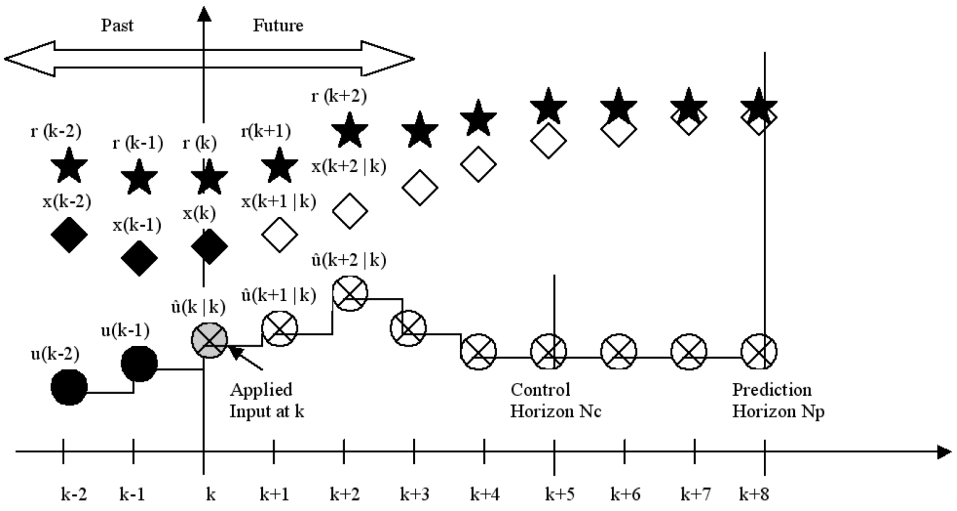

Figure 5: Principle of Predictive Control. The reference is given by r, the state of the system by x, and the control input by u. Only $\hat{u}(k \mid k)$ is implemented at instant k. The filled markers are the values known at instant k. [Niederberger, 2002]

state must be either measured or estimated to perform the estimation. These state measurements (or estimates) enter the system via the initial conditions of the cost function at the sampling instants [Allgöwer et al., 2004]; this is the way the feedback is done.

Going back to the cost function presented and assuming the whole state is perfectly measured, it can be seen that to minimize this cost function at a certain instant k, N_p future outputs are required. To calculate them, in general, a certain number of past inputs and outputs are needed. This number depends on the model used to represent the process. If we consider, for instance, the particular case of a FIR model of order n such as equation (15), an output $y(k+j)$ can be calculated only using its n previous inputs and not its previous states or outputs [Nikolaou, 2001].

$$y(k) = \sum_{i=1}^{n} h_i * u(k-i) \tag{15}$$

Depending on which of the N_p outputs we are predicting, its previous inputs and outputs will be either known (inputs up to instant $k-1$; outputs up to instant k) or predicted (inputs after instant $k-1$; outputs after instant k).

These predicted (future) inputs for instants k to $k+N_c$ (the control horizon) are the result of the optimization problem, which tries to get the predicted outputs close to the reference and normally tries to minimize the control efforts $\Delta\hat{u}(k+i)$. If the cost function is quadratic, the model is linear and there are no constraints, an explicit

solution of the problem can be found; otherwise, an iterative optimization approach has to be applied.

Once the inputs for the control horizon are obtained, only the first one, i.e., the one for instant k is applied. Then at the next sampling instant, $\hat{x}(k+1)$ is already known, and therefore, the whole optimization process is repeated.

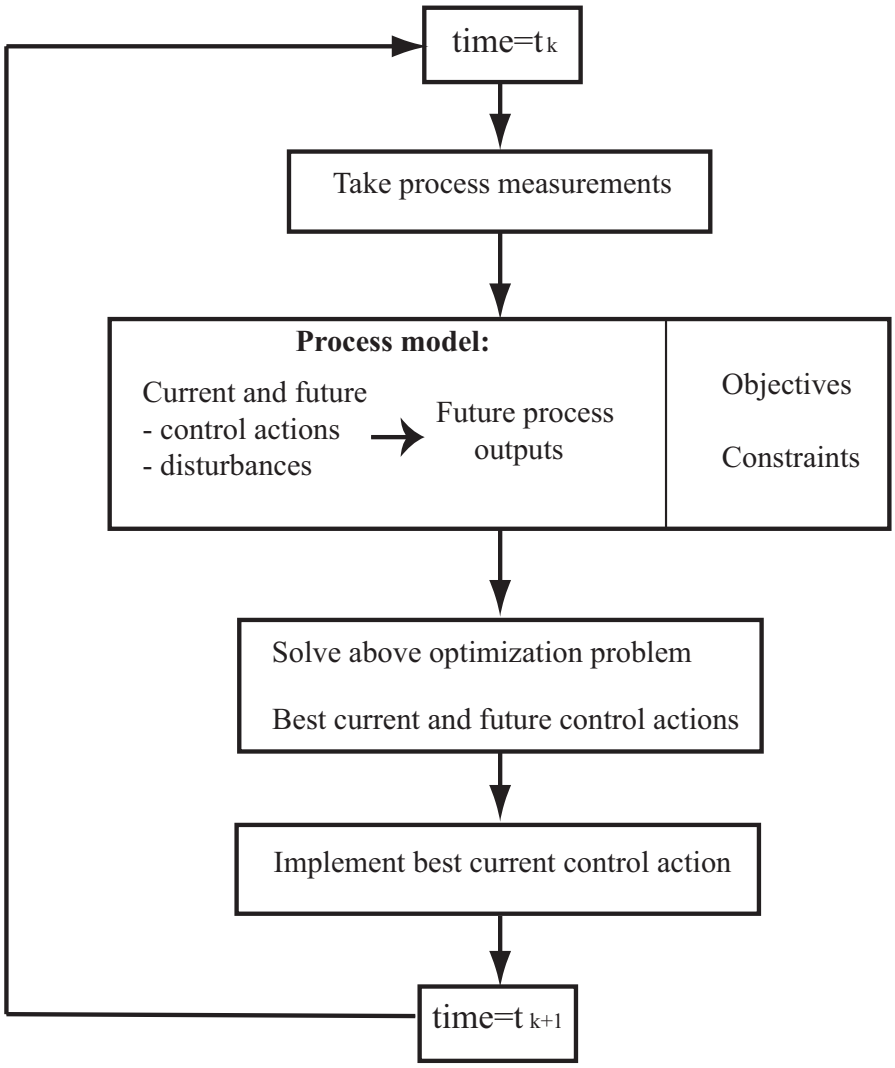

Figure 6: Scheme of the MPC strategy [Nikolaou, 2001]

3.3 Cost function choice

In the previous section, the strategy of the MPC has been explained using a very typical cost function (14), which includes the control efforts, i.e., increments in the input $\Delta u(k+j) = u(k+j) - u(k+j-1)$. Using the increments in the input in the cost function to be minimized is quite intuitive, because the aim of a controller is to make the system state reach the reference and not to need to vary the input, that is to make the increment of it equal to zero.

Even if it is not that intuitive, there is also another well-known cost function which considers directly the input, instead of its increment.

$$\mathbf{J_1} = (\mathbf{r} - \hat{\mathbf{x}})^{\mathbf{T}}\mathbf{Q}(\mathbf{r} - \hat{\mathbf{x}}) + \hat{\mathbf{u}}^{\mathbf{T}}\mathbf{\Lambda}\hat{\mathbf{u}} \tag{16}$$

where

$$\mathbf{r} = \begin{pmatrix} r(k+1) \\ \cdots \\ r(k+N_p) \end{pmatrix} \in \Re^{nN_p}$$

$$\hat{\mathbf{u}} = \begin{pmatrix} \hat{u}(k) \\ \hat{u}(k+1) \\ \cdots \\ \hat{u}(k+N_c) \end{pmatrix} \in \Re^{m(N_c+1)}$$

$$\hat{\mathbf{x}} = \begin{pmatrix} \hat{x}(k+1) \\ \cdots \\ \hat{x}(k+N_p) \end{pmatrix} \in \Re^{nN_p}$$

$\mathbf{r}$ is the vector that contains the reference points, $\hat{\mathbf{u}}$ is the input vector, $\mathbf{Q}$ and $\mathbf{\Lambda}$ are symmetric and positive definite weighting matrices.

Each of the two cost functions presented so far, (14) and (96), has an associated control algorithm. The algorithm associated to (14) is more complicated to implement than the one for (96).

To decide which algorithm to use, it has also been taken into account the fact that both control algorithms have been simulated and implemented experimentally using the 'Khepera' robot by [Niederberger, 2002] and that the results he obtained, show that both controllers have similar performances in terms of accuracy.

Consequently, the cost function in (96) is the one which has been used both for simulation and for experimental implementation.

3.4 Control algorithm

As it is mentioned in the previous section, the control algorithm depends on the cost function chosen. The one that this project uses for the MPC part is the one in (96).

When cost function (14) is used, the control algorithm is the one derived in appendix B.2.

The control algorithm developed in this section is valid specifically for the case when a linear model is chosen to represent the system, either if it is time invariant or time varying. This is the form of the model:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(\mathbf{k}) \tag{17}$$

where $\mathbf{A}$ and $\mathbf{B}$ are constants in time or not (in this last case, they should be properly written in the form $\mathbf{A}(k)$ and $\mathbf{B}(k)$).

Using the previous model, a predictor in the form

$$\hat{\mathbf{x}} = \mathbf{Z}x(k) + \mathbf{F}u(k-1) \tag{18}$$

can be derived, where $\mathbf{Z}$ and $\mathbf{F}$ are defined in appendix B.1.

Given this predictor, the solution of the optimization problem, i.e., the input vector $\hat{\mathbf{u}}$ which minimizes the given cost function in equation (96) is given by [Niederberger, 2002]:

$$\hat{\mathbf{u}} = \mathbf{K}(\mathbf{r} - \hat{\mathbf{x}}^{\mathbf{0}}) \tag{19a}$$

where

$$\mathbf{K} = (\mathbf{F}^{\mathbf{T}}\mathbf{Q}\mathbf{F} + \mathbf{\Lambda})^{-\mathbf{1}}\mathbf{F}^{\mathbf{T}}\mathbf{Q} \in \Re^{m(N_c+1)\times nN_p} \tag{19b}$$

$$\hat{\mathbf{x}}^{\mathbf{0}} = \mathbf{Z}x(k) \in \Re^{nN_p} \tag{19c}$$

Only the first line of $\hat{\mathbf{u}}$ is needed, so the first line of $\mathbf{K}$ is extracted, which is called $\mathbf{k}^{\mathbf{T}}$.

$$\mathbf{K} = [\frac{\mathbf{k}^T}{\mathbf{K}''}] \text{ where } \mathbf{k}^T \in \Re^{m\times nN_p}$$

To extract it, the following multiplication is done

$$\mathbf{k}^T = [\mathbf{I}_{m\times m} \mid \mathbf{0}_{m\times mN_c}]\mathbf{K}$$

Finally, the only input which is applied is given by

$$u(k) = \mathbf{k}^{\mathbf{T}}(\mathbf{r} - \hat{\mathbf{x}}^{\mathbf{0}}) \tag{20}$$

3.5 MPC algorithm used for the Khepera robot

As it has already been mentioned, the derived control algorithm presented in the previous section is valid when using linear models to represent the system, either for time invariant or time varying ones.

In section 2 two models were presented as well as the reasons to finally use (2) restated here

$$x(k+1) = x(k) + hv(k)\cos\theta(k) \tag{21a}$$

$$y(k+1) = y(k) + hv(k)\sin\theta(k) \tag{21b}$$

$$\theta(k+1) = \theta(k) + h\omega(k) \tag{21c}$$

where h is the sampling period.

Or its equivalent form as in equation (8)

$$\mathbf{x}_{xy\theta}(k+1) = \mathbf{x}_{xy\theta}(k) + \mathbf{B}(\theta(k))\mathbf{u}(k) \tag{22}$$

It has also been mentioned that this model is **non-linear**, **causal** and **time invariant** (see section 2). Therefore, some strategy has to be taken in order to use the control algorithm previously explained (section 3.4), which is valid for **linear** systems.

The approach proposed by [Niederberger, 2002] is the one that follows:

To substitute the angles $\theta(k)$ in the matrix $\mathbf{B}(\theta(k))$ by the reference angles $\theta_{ref}(k)$ taken from the reference trajectory. Hence the matrix

$$\mathbf{B}(\theta_{ref}(k)) = \begin{pmatrix} hcos\theta_{ref}(k) & 0 \\ hsin\theta_{ref}(k) & 0 \\ 0 & h \end{pmatrix} \tag{23}$$

$$\tag{24}$$

is used instead of $\mathbf{B}(\theta(k))$ in equation (22)

$$\mathbf{x}(k+1) = \mathbf{x}(k) + \mathbf{B}(\theta_{ref}(k))\mathbf{u}(k) \tag{25}$$

This approximation of (22) into (25) is acceptable when the real heading angle at a determined instant is close to the reference one.

The obtained model (25) is now **linear**, but **time varying**, because for each instant there is a different matrix $\mathbf{B}(\theta_{ref}(k))$. Therefore, the control algorithm can be applied now. One thing that might be interesting to mention is that for this case, where $\mathbf{B}(\theta_{ref}(k))$ is varying for every instant the optimization is done, the control algorithm (19) will be varying as well, even if its structure remains constant.

The solution of the optimization problem using the cost function (96) here repeated

$$\mathbf{J_1} = (\mathbf{r} - \hat{\mathbf{x}})^{\mathbf{T}}\mathbf{Q}(\mathbf{r} - \hat{\mathbf{x}}) + \hat{\mathbf{u}}^{\mathbf{T}}\mathbf{\Lambda}\hat{\mathbf{u}} \tag{26}$$

is the one given by (19), adapted for a varying $\mathbf{B}(\theta(k))$. That is

$$\hat{\mathbf{u}} = \mathbf{K}(\theta_{ref})(\mathbf{r} - \hat{\mathbf{x}}^{\mathbf{0}}) \tag{27a}$$

where

$$\mathbf{K}(\theta_{ref}) = (\mathbf{F}(\theta_{ref})^T\mathbf{Q}\mathbf{F}(\theta_{ref}) + \Lambda)^{-1}\mathbf{F}(\theta_{ref})^T Q \in \Re^{m(N_c+1)\times nN_p} \tag{27b}$$

$$\hat{\mathbf{x}}^{\mathbf{0}} = \mathbf{Z}x(k) \in \Re^{nN_p} \tag{27c}$$

Once again, since only the first component of the obtained input vector $\hat{\mathbf{u}}$ is applied, the next manipulation needs to be done in order to get the input which is actually implemented: a part of $\mathbf{K}(\theta_{ref}(k))$ is extracted

$$\mathbf{k}^T(\theta_{ref}) = [\mathbf{I}_{m\times m} \mid \mathbf{0}_{m\times mN_c}]\mathbf{K}(\theta_{ref})$$

and used to get the input applied to the robot as follows

$$u(k) = \mathbf{k^T}(\theta_{ref})(\mathbf{r} - \hat{\mathbf{x}}^{\mathbf{0}}(\theta_{ref})) \tag{28}$$

4 Estimator Based Control Using Flatness

As mentioned before, the control part of this project is divided in two: Model Predictive Control and Estimator Based Control Using Flatness.

To introduce this second part, the definition of **flatness** [D. Buccieri et al., 2005] is a good start.

4.1 Flatness

Definition 1 *A system* $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ *with* $\mathbf{u} \in \mathbb{R}^m$ *and* $\mathbf{x} \in \mathbb{R}^n$ *is said to be flat if there exists an output* $\mathbf{y} \in \mathbb{R}^m$ *such that:*

1. *the components of* $\mathbf{y}$ *are independent*
2. $\mathbf{x}$ *and* $\mathbf{u}$ *can be expressed as functions of* y *and its derivatives up to the p-th order*
$$\mathbf{x} = \phi(\mathbf{y}, ..., \mathbf{y}^{(p-1)}) \quad u = \psi(\mathbf{y}, ..., \mathbf{y}^{(p)}) \quad p \in \mathbb{N}$$
with ϕ *and* ψ *satisfying identically* $\dot{\phi} = f(\phi, \psi)$.

In Section 2, a continuous model for the system was presented (1). This is the model that will be used for the Estimator Based Control Using Flatness. That is why it is restated here:

$$\begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{pmatrix} = \begin{pmatrix} cos\theta(t) & 0 \\ sin\theta(t) & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v(t) \\ \omega(t) \end{pmatrix} \tag{29}$$

The state of the system the vector is

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{30}$$

where x, y and θ are the coordinates of the system at one instant.

The input of the system is

$$\mathbf{u} = \begin{pmatrix} v \\ \omega \end{pmatrix} \tag{31}$$

where v is the linear speed and ω is the angular speed of the system.

Assumption: model (29) is assumed to be an accurate representation of the system. Therefore, the term system is applied as synonym of model (29).

Note: be aware of the difference between x and $\mathbf{x}$ or y and $\mathbf{y}$. Notice as well that the components of the vectors previously presented are continuous functions of time.

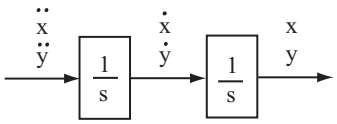

Figure 7: Chain of two integrators.

For this system, which is **flat**, we can choose an output which verifies the conditions in $DefinitionI$. Let us choose the output

$$\mathbf{y} = \begin{pmatrix} x \\ y \end{pmatrix} \tag{32}$$

To confirm the chosen flat output verifies the part of point (2) of $DefinitionI$ referring to the state, only θ needs to be written as an expression of $\mathbf{y}, ..., \mathbf{y}^{(p-1)}$components. We have

$$\theta = arctan(\frac{y}{x}) \tag{33}$$

Now, to confirm the chosen flat output verifies the part of point (2) of $DefinitionI$ referring to the input, we see that the inputs can be written as functions of $\mathbf{y}, ..., \mathbf{y}^{(p)}$ components. We have

$$v = \sqrt{\dot{x}^2 + \dot{y}^2} \tag{34a}$$

$$\omega = \frac{-\ddot{x}\dot{y} + \dot{x}\ddot{y}}{\dot{x}^2 + \dot{y}^2} \tag{34b}$$

These expressions for the input and state, give the value of $p = 2$ according to $DefinitionI$.

Note: the expressions showed before are the consequences of having a flat system, however they are not enough to prove the flatness of the system. Point (1) of $DefinitionI$ and last part of point (2) ($\mathcal{F}$ and $\mathcal{P}$ satisfying identically $\dot{\phi} = f(\phi, \psi)$) have not been proved.

A flat system can be considered as a chain of m integrators, where m is the number of inputs of the system. Thus, this representation of the system is linear.

Looking at the system as this chain of integrators, it is **linear** and it can be represented by the model

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \ddot{x} \\ \ddot{y} \end{pmatrix} \tag{35a}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix} \tag{35b}$$

In matrix form it is

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{v} \tag{36a}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} \tag{36b}$$

where

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix} \tag{37a}$$
$$\mathbf{v} = \begin{pmatrix} \ddot{x} \\ \ddot{y} \end{pmatrix} \tag{37b}$$
$$\mathbf{y} = \begin{pmatrix} x \\ y \end{pmatrix} \tag{37c}$$
$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{37d}$$
$$\mathbf{B} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{37e}$$
$$\mathbf{C} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{37f}$$

Note: Do not confuse the input of the chain of integrators $\mathbf{v} = [\ddot{x} \ \ddot{y}]'$ and the input of the real system $\mathbf{u} = [v \ \omega]'$, i.e., the linear and angular speeds.

We can observe that the model is **decoupled**. This implies we can see it as m chains of p integrators, i.e., two chains of two integrators.

In this case, the model is represented by

$$\dot{\mathbf{x}} = \mathbf{A_1}\mathbf{x} + \mathbf{B_1}\mathbf{v_2} \tag{38a}$$
$$\mathbf{z_1} = \mathbf{C_1}\mathbf{x} \tag{38b}$$
$$\dot{\mathbf{y}} = \mathbf{A_2}\mathbf{y} + \mathbf{B_2}\mathbf{v_1} \tag{38c}$$
$$\mathbf{z_2} = \mathbf{C_2}\mathbf{y} \tag{38d}$$

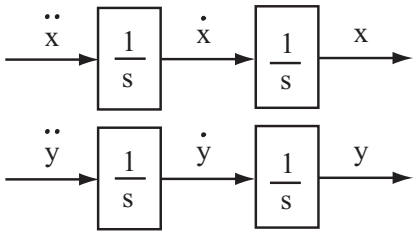

Figure 8: Two chain of two integrators, since the model is decoupled.

where

$$\mathbf{x} = \begin{pmatrix} x \\ \dot{x} \end{pmatrix} \tag{39a}$$

$$\mathbf{y} = \begin{pmatrix} y \\ \dot{y} \end{pmatrix} \tag{39b}$$

$$\mathbf{v_1} = \ddot{x} \tag{39c}$$

$$\mathbf{v_2} = \ddot{y} \tag{39d}$$

$$\mathbf{z_1} = x \tag{39e}$$

$$\mathbf{z_2} = y \tag{39f}$$

$$\mathbf{A_1} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \tag{39g}$$

$$\mathbf{A_2} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \tag{39h}$$

$$\mathbf{B_1} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{39i}$$

$$\mathbf{B_2} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{39j}$$

$$\mathbf{C_1} = \begin{pmatrix} 1 & 0 \end{pmatrix} \tag{39k}$$

$$\mathbf{C_2} = \begin{pmatrix} 1 & 0 \end{pmatrix} \tag{39l}$$

Model (36), i.e. the one considering the whole system, is the one used in order to explain the theory in this section to make it a bit simpler. On the other hand, model (38), i.e. the decoupled one, is the one used both in simulation and implementation, because it allows to apply different controls for the coordinates x and y.

4.2 EBCUF general strategy

The change of reference that converts our system into a chain of integrators (see section 4.1), allows us to see it as a **linear** system. Hence, **Linear Control Theory**

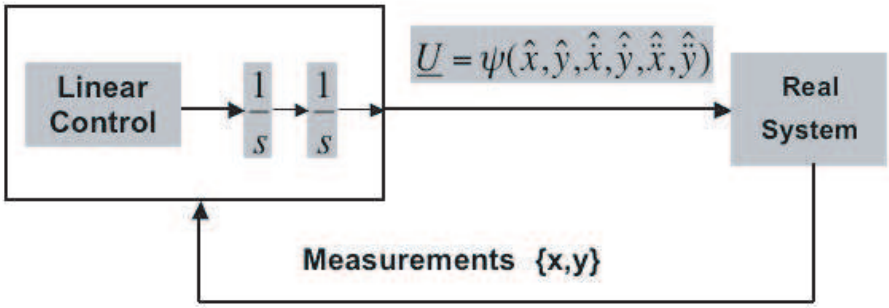

Figure 9: EBCUF Strategy. Relation between the chain of integrators and the real system, by using the flatness property to get the input of the real system.

can be applied to this linear chain of integrators. Then using the property of flatness (34), the input applied to the real system can be obtained in terms of the flat output, and its first and second derivatives, which are contained in the state and the state derivatives of the chain of integrators as it is shown in equation (34).

The control of the chain of integrators is done by using **state feedback**, i.e. the state, instead of the output, is used for control. This state feedback is done with a **closed-loop state estimator**, i.e. an estimator which uses not only the inputs of the system but also the outputs in order to estimate the state [Chi-Tsong Chen, 1999].

Hence, for the control of the chain of integrators there will be a **state estimator** and a classical **controller**.

4.2.1 State estimator

The state estimator is introduced for control when a state feed-back is desired and not all the state variables are accessible.

The aim of the estimator is to get the values as accurate as possible for the state components presented in (37). The notation of the state and its derivative is

$$\hat{\mathbf{x}} = \begin{pmatrix} \hat{x} \\ \hat{y} \\ \dot{\hat{x}} \\ \dot{\hat{y}} \end{pmatrix} \tag{40a}$$

$$\dot{\hat{\mathbf{x}}} = \begin{pmatrix} \dot{\hat{x}} \\ \dot{\hat{y}} \\ \ddot{\hat{x}} \\ \ddot{\hat{y}} \end{pmatrix} \tag{40b}$$

A state estimator can be either an open-loop or a closed-loop estimator. As usual, close-loop provides more satisfactory results. The difference between open and

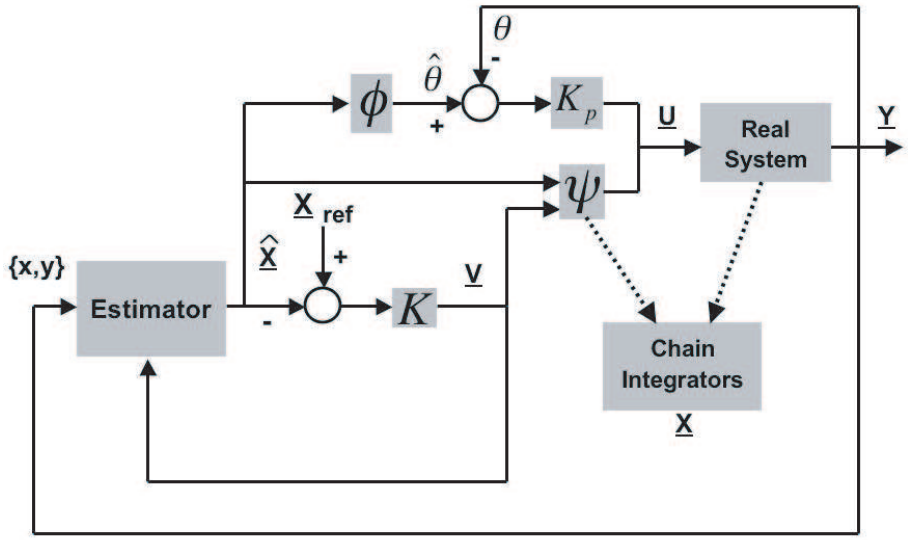

Figure 10: Detail of the control of the chain of integrators using an estimator

closed-loop is that the first one does not use the output values to estimate the state while the second one does use them.

The state estimator chosen in this project is the Luenberger estimator or observer, the equation of which is given by [Chi-Tsong Chen, 1999]

$$\dot{\hat{\mathbf{x}}} = (\mathbf{A} - \mathbf{LC})\hat{\mathbf{x}} + \mathbf{Bv} + \mathbf{Ly} \tag{41}$$

where $\dot{\hat{\mathbf{x}}}$ and $\hat{\mathbf{x}}$ are given by (40); $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{v}$ and $\mathbf{y}$ are given by (37) and $\mathbf{L}$ is a matrix found by pole placement of $(\mathbf{A}-\mathbf{LC})$. The sizes of these vectors and matrices are:

$$\begin{gathered}\hat{\mathbf{x}} \in \mathbb{R}^n \\ \dot{\hat{\mathbf{x}}} \in \mathbb{R}^n \\ \mathbf{A} \in \mathbb{R}^{n\times n} \\ \mathbf{B} \in \mathbb{R}^{n\times m} \\ \mathbf{C} \in \mathbb{R}^{m\times n} \\ \mathbf{v} \in \mathbb{R}^m \\ \mathbf{y} \in \mathbb{R}^m \\ \mathbf{L} \in \mathbb{R}^{n\times m}\end{gathered}$$

The reason to calculate $\mathbf{L}$ by placing the poles of $(\mathbf{A} - \mathbf{LC})$ is explained in the next lines.

The error between the actual state and the estimated one is [Chi-Tsong Chen, 1999]

$$\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}} \tag{42}$$

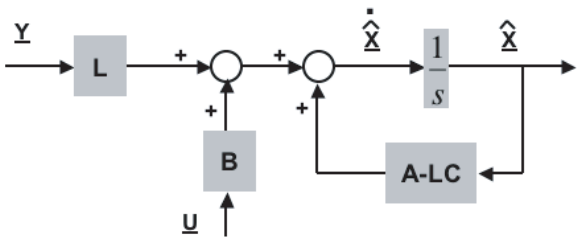

Figure 11: Luenberger estimator: block diagram.

the derivative of this error is

$$\dot{\mathbf{e}} = \dot{\mathbf{x}} - \dot{\hat{\mathbf{x}}} \tag{43}$$

$$\dot{\mathbf{e}} = (\mathbf{A} - \mathbf{LC})\mathbf{x} - (\mathbf{A} - \mathbf{LC})\hat{\mathbf{x}} \tag{44}$$

$$\dot{\mathbf{e}} = (\mathbf{A} - \mathbf{LC})\mathbf{e} \tag{45}$$

The eigenvalues of $(\mathbf{A} - \mathbf{LC})$ can be placed arbitrarily. It allows to control the error. Hence, it is possible to control the error to approach zero or, equivalently, for the estimated state to approach the actual state. It is possible by choosing negative eigenvalues for $(\mathbf{A} - \mathbf{LC})$. One option to do this is using Ackermann's formula.

4.2.2 Control

The controller consists on a constant matrix $\mathbf{K} \in \mathbb{R}^{m \times n}$ which works as a proportional controller that uses the error between the reference state $\mathbf{x_{ref}}$ and the estimated one $\hat{\mathbf{x}}$. The output of the controller gives the input for the chain of integrators, that is

$$\mathbf{v} = \begin{pmatrix} \ddot{\hat{x}} \\ \ddot{\hat{y}} \end{pmatrix} \tag{46}$$

This input is **not** applied to the chain of integrators because it is a kind of **'virtual system'** which has been used to apply linear control. In the **real system**, we need to apply the input $\mathbf{u} = [v \ \ \omega]'$, which is found by using flatness (34).

The controller is

$$\mathbf{v} = \mathbf{K}(\mathbf{x_{ref}} - \hat{\mathbf{x}}) \tag{47}$$

By placing the poles of $(\mathbf{A} - \mathbf{BK})$, the matrix $\mathbf{K}$ is determined. Once again, the eigenvalues of $(\mathbf{A} - \mathbf{BK})$ are chosen to be negative.

Apart from the $\mathbf{K}$ controller with which $\ddot{\hat{x}}$ and $\ddot{\hat{y}}$ (needed to apply (34) and for the estimator) are obtained, the control is done with a **proportional controller** K_p, that controls the orientation θ of the system. This proportional controller is **basic** to make the EBCUF controller be stable, because the choice of a K_p gain high enough is

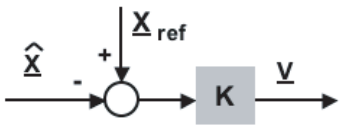

Figure 12: K controller: block diagram

what makes the real system be linear and therefore give the possibility to use linear control. This proportional controller is

$$\omega = K_p(\hat{\theta} - \theta) \tag{48}$$

where

$$\hat{\theta} = \arctan \hat{y}/\hat{x} \tag{49}$$

4.2.3 EBCUF global effect

We have said that the EBCUF is formed by a state estimator **L** and the control applied with **K** and K_p.

The equation of the estimator **L** and the **K** part of the control is

$$\dot{\hat{\mathbf{x}}} = \mathbf{Ax} + \mathbf{BK}(\mathbf{x_{ref}} - \hat{\mathbf{x}}) + \mathbf{LC}(\mathbf{x} - \hat{\mathbf{x}}) \tag{50}$$

We have also mentioned that:

- **L** is chosen for the estimated state $\hat{\mathbf{x}}$ to approach the actual state **x**
- **K** is chosen for the estimated state $\hat{\mathbf{x}}$ to approach the reference state $\mathbf{x_{ref}}$

Hence, the Estimator Based Control Using Flatness, by the separation property makes the actual state **x** approach $\mathbf{x_{ref}}$. Since this property is valid for linear systems, we need K_p to be high enough to make the real system become linear, as it is shown in section 4.3.

4.3 Preliminaries for the proof of stability

This section presents how a K_p high enough makes the real system become linear and therefore allow the use of linear control, that is why it has been included. However, this project has **not** properly worked on the stability of this control strategy. The equations here presented are part of [D. Buccieri et al., 2005].

The idea is to prove that x, y converge to zero. Bringing θ to zero is not very important at first, since once the system is on $x = 0$ and $y = 0$, θ can be controlled without any problem.

Using the control laws in previously developed in a particular way (50), (34a) and (48), the closed-loop system (system + estimator) can be written:

$$\dot{\hat{x}} = A\hat{x} + BK(x_{ref} - \hat{x}) + LC(x - \hat{x}) \tag{51}$$

$$\dot{x} = \sqrt{\dot{x}^2 + \dot{y}^2}\cos\theta \tag{52}$$

$$\dot{y} = \sqrt{\dot{x}^2 + \dot{y}^2}\sin\theta \tag{53}$$

$$\dot{\theta} = K_p(\hat{\theta} - \theta) \tag{54}$$

Now (54) can be write:

$$\frac{\dot{\theta}}{K_p} = \hat{\theta} - \theta \tag{55}$$

If $K_p \gg 0$, (55) can be considered as:

$$0 = \hat{\theta} - \theta \tag{56}$$

Then using (56) and (49)

$$\theta = \hat{\theta} = \arctan(\frac{\hat{\dot{y}}}{\hat{\dot{x}}}) \tag{57}$$

Substituting (57) into (52) and 53, the result is:

$$\dot{x} = \sqrt{\dot{x}^2 + \dot{y}^2}\cos(\arctan(\frac{\hat{\dot{y}}}{\hat{\dot{x}}})) = \hat{\dot{x}} \tag{58}$$

$$\dot{y} = \sqrt{\dot{x}^2 + \dot{y}^2}\sin(\arctan(\frac{\hat{\dot{y}}}{\hat{\dot{x}}})) = \hat{\dot{y}} \tag{59}$$

Then the closed-loop system (51)-(54) can be reduce using notation $\xi_1 = x$, $\xi_2 = y$, $\xi_3 = \hat{x}$, $\xi_4 = \hat{\dot{x}}$, $\xi_5 = \hat{y}$ and $\xi_6 = \hat{\dot{y}}$ as:

$$\begin{pmatrix} \dot{\xi}_1 \\ \dot{\xi}_2 \\ \dot{\xi}_3 \\ \dot{\xi}_4 \\ \dot{\xi}_5 \\ \dot{\xi}_6 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ l_1 & 0 & -l_1 & 1 & 0 & 0 \\ l_2 & 0 & -k_1 - l_2 & -k_2 & 0 & 0 \\ 0 & l_1 & 0 & 0 & -l1 & 1 \\ 0 & l_2 & 0 & 0 & -k_1 - l_2 & -k_2 \end{pmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \xi_4 \\ \xi_5 \\ \xi_6 \end{pmatrix}, \tag{60}$$

where $K = \begin{pmatrix} k_1 & k_2 & 0 & 0 \\ 0 & 0 & k_1 & k_2 \end{pmatrix}$ and $L = \begin{pmatrix} l_1 & l_2 & 0 & 0 \\ 0 & 0 & l_1 & l_2 \end{pmatrix}^T$. The system (60) is linear with the eigenvalues:

$$\lambda_{1,2} = -l1 \tag{61}$$

$$\lambda_{3,4} = \frac{1}{2}(-k_2 - \sqrt{-4k_1 + k_2^2}) \tag{62}$$

$$\lambda_{5,6} = \frac{1}{2}(-k_2 + \sqrt{-4k_1 + k_2^2}) \tag{63}$$

The reduced system is asymptotically stable since there exists a choice of values k_1, k_2, l_1, l_2 for which $\lambda_i < 0$, $\forall i = 1, \ldots, 6$.
Considering (55)

$$\frac{\dot{\theta}}{K_p} = \hat{\theta} - \theta = g(\hat{\theta}, \theta) \tag{64}$$

and the new variable y Where $h(\hat{\theta})$ are the root of $g(\hat{\theta}, \theta) = 0$ So

$$g(\hat{\theta}, \theta) = -y \tag{65}$$

The boundary-layer model $\frac{dy}{d\tau}$ is exponentially stable hence $\frac{\partial g}{\partial y}$ is hurwitz. In our case $\frac{\partial g}{\partial y} = -1$

4.4 EBCUF variations

The theoretical principles explained previously in this section are a general case. Many other options like a reduced order estimator or adding an extra proportional controller can also be considered.

In this project three different controllers, all using flatness, were tested. Here, the features that define the way they work are presented and the conclusions are drawn in the section of simulations (9).

4.4.1 Case A

The **features** of this controller are:

- Components of the state of the real system controlled: x and y.
- Measurements taken from the real system: x and y.
- Calculation of the real input v: applying flatness (34a).
- Calculation of the real input ω: applying flatness (34b).
- Estimators state: $\hat{x}, \hat{\dot{x}}, \hat{y}$ and $\hat{\dot{y}}$.

The controller is designed applying directly the theory previously explained. Therefore there is a state estimator as in section 4.2.1 and a controller as in 4.2.2.

Remember that the state of the real system has three components x, y and θ. With this controller we control x and y, but not θ, which is something not very desirable.

However, with no perturbations, if the estimator is given the initial angle θ, it would be possible to do an open loop control of θ. The way to give this initial orientation is through the initial values of the components $\hat{\dot{x}}$ and $\hat{\dot{y}}$ of the estimator state. This is based on the fact that

$$\theta = \arctan \dot{y}/\dot{x} \tag{66}$$

For instance, to introduce an initial orientation of $\theta_0 = 0$, we should assign $\hat{\dot{x}} = constant$ and $\hat{\dot{y}} = 0$ to these components of the estimator.

4.4.2 Case B

The **features** of this controller are:

- Components of the state of the real system controlled: x, y and θ.
- Measurements taken from the real system: x, y and θ.
- Calculation of the real input v: applying flatness (34a).
- Calculation of the real input ω: using a proportional controller (Kp).
- Estimators state: $\hat{x}, \hat{\dot{x}}, \hat{y}$ and $\hat{\dot{y}}$.

The structure of this controller is equivalent to the **case A** for the linear speed, v, which is calculated by applying (34a).

What is added in this case is the proportional controller previously mentioned which has the orientation error, i.e., $(\hat{\theta} - \theta)$, as input, where $\hat{\theta}$ is computed from $\hat{\dot{x}}$ and $\hat{\dot{y}}$. The output of this controller is directly ω. The equations for the controller are:

$$\hat{\theta} = \arctan \hat{\dot{y}}/\hat{\dot{x}} \tag{67}$$

$$\omega = K_p(\hat{\theta} - \theta) \tag{68}$$

The advantage of this controller is that it is controlling θ as well, which is something that makes it more robust.

4.4.3 Case C

The **features** of this controller are:

- Components of the state of the real system controlled: x, y and θ.
- Measurements taken from the real system: x, y and θ.
- Calculation of the real input v: applying flatness (34a).
- Calculation of the real input ω: applying flatness (34b) and also using a proportional controller (Kp).
- Estimators state: $\hat{x}, \hat{\dot{x}}, \hat{y}$ and $\hat{\dot{y}}$.

This option combines case A and case B. That means that ω is calculated applying both flatness and a proportional controller. First, equation (34b) is used to find ω and then an extra term is added to it, which is the error $(\hat{\theta} - \theta)$ scaled by a constant, K_p. Doing this, θ is controlled as well as x and y.

Therefore, the computation of the input ω is done with the following equation

$$\omega = \frac{-\hat{\ddot{x}}\hat{\dot{y}} + \hat{\dot{x}}\hat{\ddot{y}}}{\hat{\dot{x}}^2 + \hat{\dot{y}}^2} + K_p(\hat{\theta} - \theta) \tag{69}$$

4.4.4 Case D

The **features** of this controller are:

- Components of the state of the real system controlled: x, y and θ.
- Measurements taken from the real system: x, y and θ.
- Calculation of the real input v: applying flatness (34a).
- Calculation of the real input ω: using a proportional controller (Kp).
- Estimators state: $\dot{x}$ and $\dot{y}$.

The main difference between this controller and the others is that this one is a reduced order controller. Hence, the coordinates x and y are no longer estimated, but used directly from the measurements. Then, only $\dot{x}$ and $\dot{y}$ are estimated.

5 Position estimation

When it comes to implement a closed loop control of a system it is important to have an accurate state or output estimation. In this project, the two controllers that have been used need feedback of the estate, i.e.: x, y and θ. In order to get these position coordinates, the follow steps have been developed:

- Acquisition of measurements of the state using an image treatment algorithm
- Filtering of the measurements with a Kalman Filter

For the acquisition of the measurements, the image treatment used in this project is based on the algorithms proposed by [Herrera, 1999] and the VIs developed in [J.M. Miamo ,2005]

The first thing to do to get the state measures is the acquisition of the image obtained with a camera, which is done with a toolbox made by Christoph Salzmann. Then, the image treatment algorithm proposed by [Herrera, 1999] and here explained is used.

Once the image acquisition is done, this image is converted into a matrix of pixels which contain the color (which is defined with 8 bits, so it is a number from 0 = white to 256 = black) of the small part of the image represented by the pixel.

After that, the matrix with different colors is converted to a black and white one using the **thresholding** process. This process consists on making light colors become white and dark colors become black. The decision is made after fixing a number. Then, colors with a number higher than the fixed one will be black and colors with a number lower than the fixed one will be white.

This is done in order to find the robot's position. And it is helpful to do it this way because the robot has been painted in white, with two black points on it and it is moving on a black surface.

Figure 13: Black and white platform used for the experiments.

Once we have the black and white image, the idea is to get the center of mass of the white points, which is supposed to be the center of the robot. To do this, three steps are done, in order to minimize time and to reduce noise.

1. First, the entire area in the image is considered, but only one out of 25 points is considered in the matrix. These points that have been chosen are equidistant, i.e., horizontal and vertical divisions every 5 pixels are done. With the points taken into account, a center of mass of the white points is calculated.

2. A smaller region around the previously calculated center of mass is taken into account. In this region, one out of 3 points is considered. With these points, a new center of mass is calculated.

3. Finally, a smaller region around the second center of mass found is considered. In this case, all the points in the region are taken into account. This third center of mass is considered to be the position, i.e., x and y coordinates of the robot.

Then, the orientation needs to be found. That is why two black circles are painted on the robot. Differently from what [Herrera, 1999] did, these two circles have different sizes, so that the direction is uniquely found from 0 and 2π.

To find these two circles, the white area, which is the robot, is considered. Then a vertical line is done and the center of mass of black points in each of the parts, right and left, is calculated. If the distance of the centers of mass is close to this vertical line, an horizontal division is done and the centers of mass of the black points in the top and bottom parts are calculated again. The line given by these two centers of mass gives two possible orientations. Finally, looking at the sizes of the two black circles, the real and unique orientation is found.

6 MPC simulations in Matlab

The Model Predictive Control algorithm simulated in Matlab is the one explained in Section 3.4, which uses the cost function in (96), that is here restated:

$$\mathbf{J_1} = (\mathbf{r} - \hat{\mathbf{x}})^{\mathbf{T}}\mathbf{Q}(\mathbf{r} - \hat{\mathbf{x}}) + \hat{\mathbf{u}}^{\mathbf{T}}\mathbf{\Lambda}\hat{\mathbf{u}} \tag{70}$$

and model (2), that is

$$x(k+1) = x(k) + hv(k)\cos\theta(k) \tag{71a}$$
$$y(k+1) = y(k) + hv(k)\sin\theta(k) \tag{71b}$$
$$\theta(k+1) = \theta(k) + h\omega(k) \tag{71c}$$

where h is the sampling period.

As we see, both the model and the control algorithm are written in discrete time, and it is this way how the matlab codes have been written too.

One of the main objectives of this project has been to make it **understandable** to those who might be interested in it in the future. That is why the code of the simulation has been written with the explanations which might be necessary, and hopefully enough, to make everything clear. The expectations are that someone who reads this report will be able to understand the code and **reuse** or **'recycle'** it with no need to start the code from zero.

In this section the performance of the MPC using different parameter settings and trying several reference trajectories is presented. In all of them, a table with the parameters will be presented. Table 1 shows the parameters which have been kept constant in all simulations.

Note: The **length** and **time units** do not affect the results obtained in simulation. Hence, the units here stated are just used as a tool to give the results in a more clear way. However, for experimentation, it is necessary to adjust the needed units that imply a **feasible command** and a **possible control** of the real system, taking also into account the **state measurements acquisition speed**.

General Constants	Symbol	Value
Sampling time of system	h[ms]	10
Axle length of the robot	L[mm]	53
Number of iterations done	$runs$	1000
Initial input vector	$\mathbf{u}_{0-}$ ([mm/s], [rad/s])	[0, 0]
Input weight	Λ	I_{mxm}

Table 1: Table with the constants used for all simulations.

Note: The way to present the graphics in each section will not be standard for all of them. Depending on the case, different parts of the simulation graphics will be presented.

6.1 Effect of the matrix Q

This section tries different options for the weighting matrix Q that appears in the cost function (70), that weight the error between the actual state and the reference. In order to try different structures for the matrix Q, different cases have been simulated. The differences between them are basically based on:

- Equal or different weights for the errors in the different components of the state: $[x, y, \theta]$
- Equal or different weights for the errors in the different states considered through the prediction horizon (N_p).

Constants	Symbol	Value
Reference trajectory		sinus
Prediction horizon	N_p	6
Control horizon	N_c	5
Initial state vector ([mm],[mm],[rad])	$\mathbf{x}_0$	(60, 40, $\pi/4$)
CASE **A**: Variables	Symbol	Value
x coordinate error weight in matrix Q	Q_x	1
y coordinate error weight in matrix Q	Q_y	1
θ coordinate error weight in matrix Q	Q_θ	1
CASE **B**: Variables	Symbol	Value
x coordinate error weight in matrix Q	Q_x	100
y coordinate error weight in matrix Q	Q_y	100
θ coordinate error weight in matrix Q	Q_θ	100
CASE **C**: Variables	Symbol	Value
x coordinate error weight in matrix Q	Q_x	10000
y coordinate error weight in matrix Q	Q_y	10000
θ coordinate error weight in matrix Q	Q_θ	10000
CASE **D**: Variables	Symbol	Value
x coordinate error weight in matrix Q	Q_x	$100(Np - i)$ where $i\epsilon[0, N_p - 1]$
y coordinate error weight in matrix Q	Q_y	$100(Np - i)$ where $i\epsilon[0, N_p - 1]$
θ coordinate error weight in matrix Q	Q_θ	$100(Np - i)$ where $i\epsilon[0, N_p - 1]$
CASE **E**: Variables	Symbol	Value
x coordinate error weight in matrix Q	Q_x	50
y coordinate error weight in matrix Q	Q_y	1000
θ coordinate error weight in matrix Q	Q_θ	10

Table 2: Table with the parameters for different structures of the matrix Q.

Cases A, B and C have been simulated using constant weights. These weights remain constant for the errors in all the components of the state, $[x, y, \theta]$, and also constant for the whole prediction horizon.

Case D considers weights which are constant for the errors in all the components of the state in the same iteration, but they decrease as we go further into the prediction horizon, i.e., errors in a nearer future are given higher weights.

In case E, the weights remain constant for the whole prediction horizon; however, the errors in the three components have different weights. The weights were assigned based on the errors in previous cases. Then, the magnitudes of the three weights have been chosen directly proportional to the average errors observed. That is, since the average errors verified $e_y > e_x > e_\theta$, the weights have been chosen so that $Q_x > Q_y > Q_\theta$.

In figure 14, different views of the XY graph are shown:

- In the first graph of the figure, the whole simulation for the 5 cases is presented. However the differences are not visible.
- The second graph in the figure shows the area close to the initial conditions. It can be observed that the order of cases, from fastest to slowest to reach the reference trajectory are: case C, D, B, E and A.
- The third graph, shows a small region in the first quarter of period. In this case, the order of cases from closest to furthest from the reference trajectory is once again: C, D, B, E and A.
- The last one is around the first quarter of period, where the y coordinate reaches its maximum. Here, the order of cases from closest to furthest to the reference is: E, C, B, D and A.

Conclusions:

- Case A offers the worst performance. It is probably due to the small magnitude of its weights compared to the other cases.
- Cases B, C and D are very similar, being C always the best of them, probably to the high magnitude of its weights. However, the difference between case C and B is not very big for having weights 100 orders of magnitude bigger. Case D, which gives a higher weight to the closer instants in the prediction horizon and a lower weight to the further instants, offers an average performance in between cases B and C.
- Case E, which gives a different weight to the error in each component, but constant in all the prediction horizon, shows a not very good performance around the regions 1/5, 3/5 and 5/5 of the simulation (compared to the other cases). However, around the regions 2/5 and 4/5 of the simulation, the performance is quite good. It is due to the fact that the higher weight is given to the error in y, which is bigger around the regions 2/5 and 4/5 of the simulation as it presented in figure 15.

- Finally, it is **important** to remark that these simulations show different cases with different strategies for the choice of weights. However, the results obtained cannot be used in a general way to measure the performance of the strategies because only some magnitudes have been used, i.e., a deeper study should be done by trying many different weights for each of the strategies.

In figure 15, the errors in the three components of the state are presented.We can see the error in x is bigger in regions 1/5, 3/5 and 5/5 of the simulation; while the error in y is bigger in regions 2/5 and 4/5 of the simulation. On the other side, the tracking of θ is really good; there is an error due to the initial conditions and then around regions 2/5 and 4/5 of the simulation, which is really small except for the case A.

Also in figure 15, it can be observed a behavior which is repeated **for all simulations**. It is that the error in x changes its sign, and therefore becomes minimum, when y gets its maximum values, around $t = 2500ms$ and $t = 7500ms$. Around the same points, the error in y and θ, becomes maximum. The fact that x changes its sign, that is the real trajectory stays always either over or underneath the reference is because the offset in y maintains always its sign.

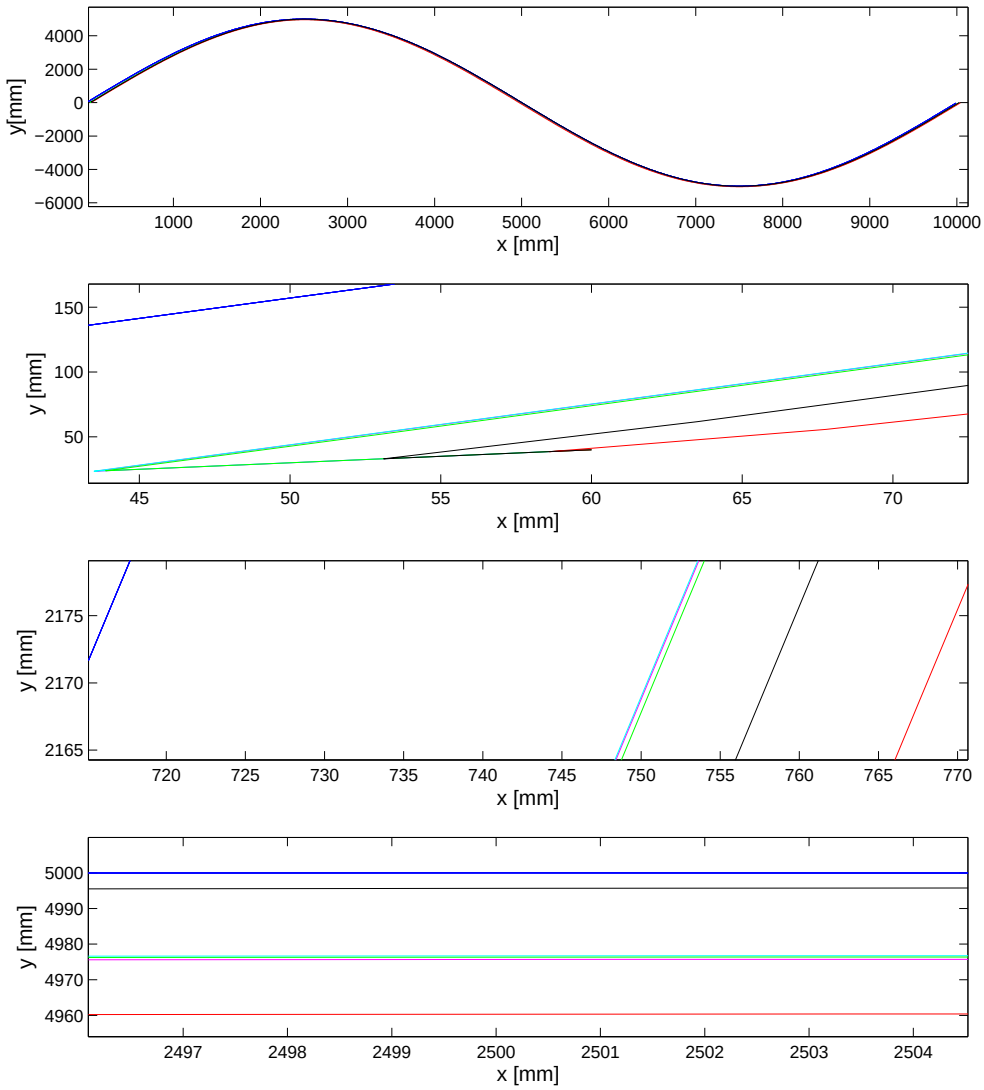

Figure 14: Simulations with different values for the weights of the Q matrix: different regions in the x-y plane.

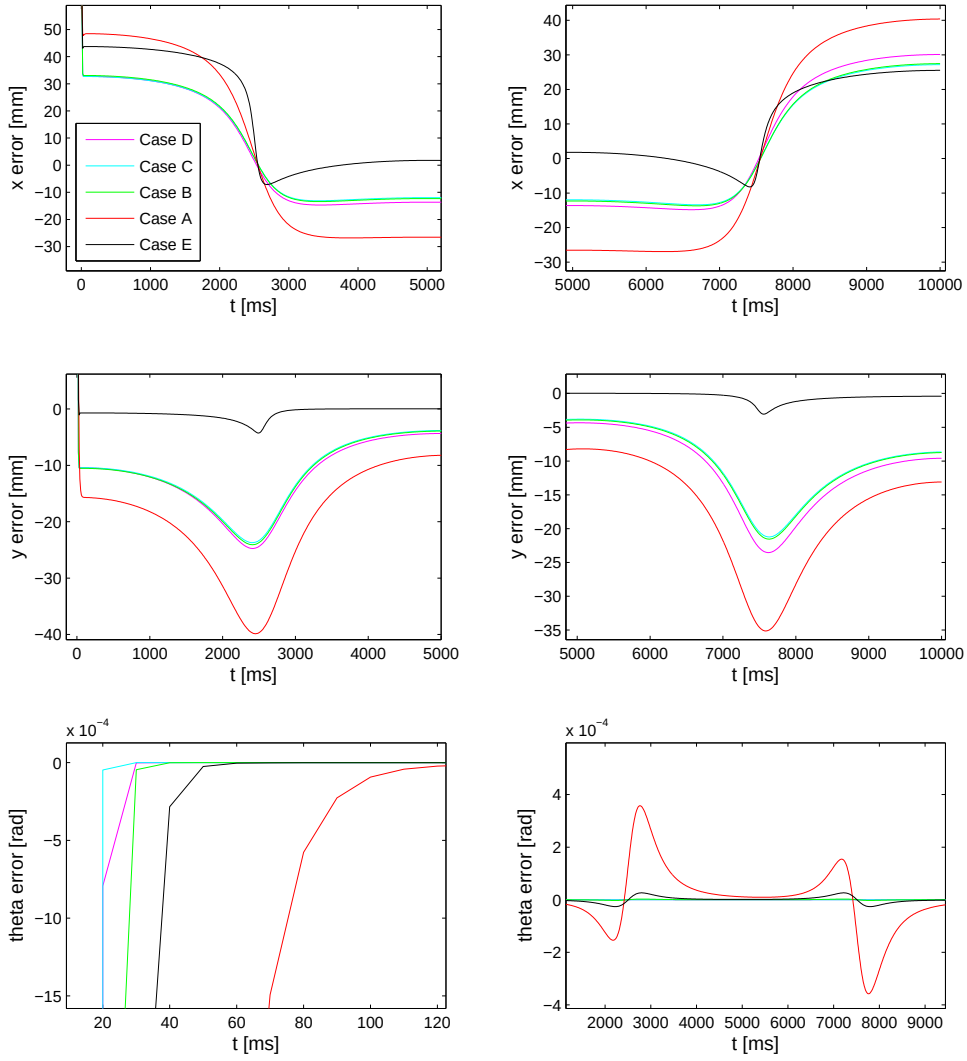

Figure 15: Simulations with different values for the weights of the Q matrix: errors in x,y and θ.

6.2 Effect of the prediction and control horizons: N_p and N_c

Constants	Symbol	Value
Reference trajectory		sinus
Initial state vector ([mm],[mm],[rad])	$\mathbf{x}_0$	(60, 40, $\pi/4$)
x coordinate error weight in matrix Q	Q_x	100
y coordinate error weight in matrix Q	Q_y	100
θ coordinate error weight in matrix Q	Q_θ	100
CASE **A**: Variables	Symbol	Value
Prediction horizon	N_p	2
Control horizon	N_c	1
CASE **B**: Variables	Symbol	Value
Prediction horizon	N_p	6
Control horizon	N_c	5
CASE **C**: Variables	Symbol	Value
Prediction horizon	N_p	10
Control horizon	N_c	9
CASE **D**: Variables	Symbol	Value
Prediction horizon	N_p	10
Control horizon	N_c	1

Table 3: Table with the parameters for different prediction and control horizons: N_p andN_c.

Cases A, B and C correspond to different levels (low, medium and high) for the prediction and control horizon, all with a difference of one between Np and Nc. Then, case D, uses the highest value used for Np and the lowest value for Nc.

The simulations show that the differences between these cases are important. Comparing to the previous section (6.1), where the effect of matrix Q was considered, the errors are quite smaller. We can see in the four graphs in figure 16, that in different regions of the simulation the differences are really small.

It can be said though that when increasing both Np and Nc, all the errors become smaller. In case D, when Np is 10 and Nc is 1, the error in x and y remains as good as in case C, however, there is a high error in θ, if we compare to the other cases. We can also say that the performances between these cases differ as the time goes on.

Once again, we can say that the error in x is higher in the regions 1/5, 3/5 and 5/5 of the simulations, while the error in y is higher in regions 2/5 and 4/5.

Conclusions:

- As the prediction and control horizons Np and Nc increase, all the errors become smaller.
- If the difference between Np and Nc increases, an error in θ appears.

- The differences between the different cases are quite small.

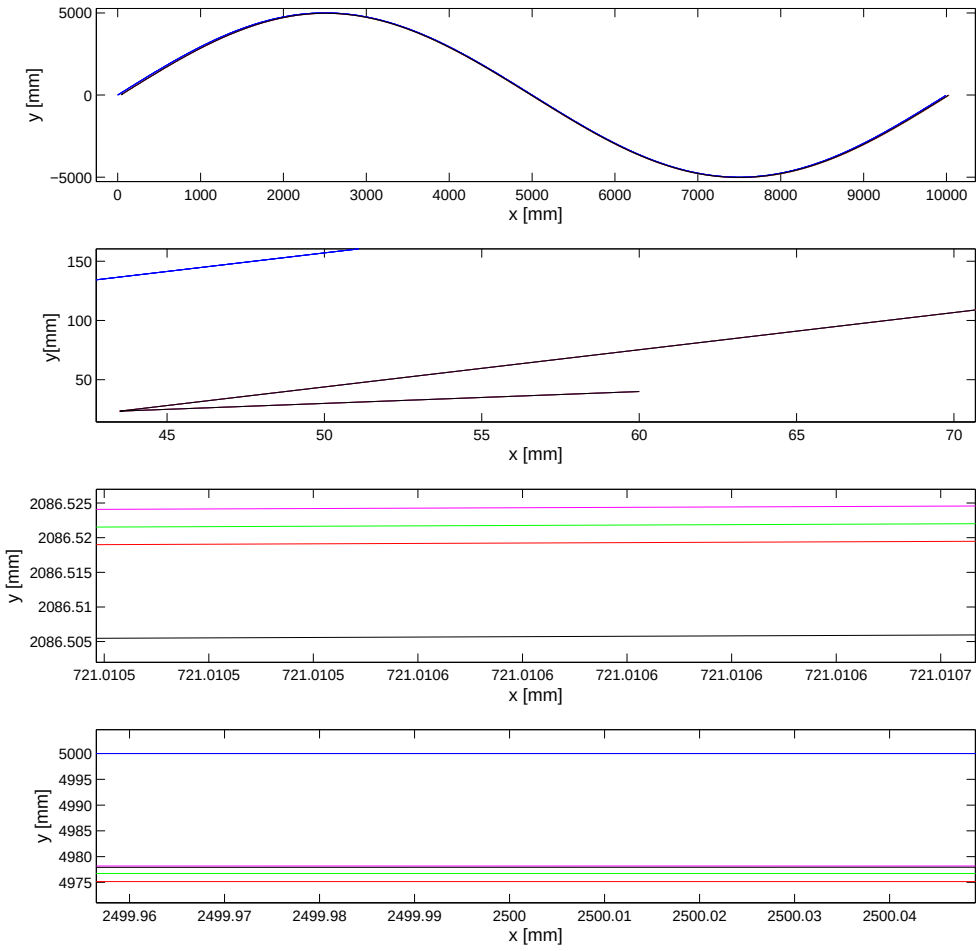

Figure 16: Simulations with different values for Np and Nc: different regions in the x-y plane.

6.3 Effect of the initial conditions: $\mathbf{x}_0$

Constants	Symbol	Value
Reference trajectory		sinus
Prediction horizon	N_p	10
Control horizon	N_c	9
x coordinate error weight in matrix Q	Q_x	10000
y coordinate error weight in matrix Q	Q_y	10000
θ coordinate error weight in matrix Q	Q_θ	10000
CASE **A**: Variables	Symbol	Value
Initial state vector ([mm],[mm],[rad])	$\mathbf{x}_0$	(60, 40, $\pi/4$)
CASE **B**: Variables	Symbol	Value
Initial state vector ([mm],[mm],[rad])	$\mathbf{x}_0$	(-40, 60, $5\pi/4$)
CASE **C**: Variables	Symbol	Value
Initial state vector ([mm],[mm],[rad])	$\mathbf{x}_0$	(-60,- 40, $-\pi/2$)
CASE **D**: Variables	Symbol	Value
Initial state vector ([mm],[mm],[rad])	$\mathbf{x}_0$	(40, -60, $\pi/2$)

Table 4: Table with the parameters for different initial conditions.

Looking at figure 19 it can be seen that the differences in the magnitudes of the errors for the different cases remain almost constant. From highest to smallest errors, the cases are ordered as follows: case C, case B, case A and case D.

The reference trajectory is a sinus, which linearized around the origin is the straight line with slope $\pi/4$. Then, the errors in θ are: 0 for case A, $\pm\pi$ for case B, $-3\pi/4$ for case C and $\pi/4$ for case D.

Figure 18 shows clearly how the error in y maintains its sign, positive or negative, i.e., the real trajectory curve stays does not cross the reference trajectory at any point.

Conclusions:

- The magnitude of the errors, both in x and y, during the whole simulation is highly related to the initial error in θ. This is caused by the approximation which is done when using $B(\theta_{ref}(k))$ instead of $B(\theta(k))$. Doing this, we are actually controlling the system as if it was decoupled and it is not. Therefore, once the initial orientation error is corrected ($\theta = \theta_{ref}$), it will not be modified even if there is an offset in x or in y.

- Once again, the offset in y, does not change its sign.

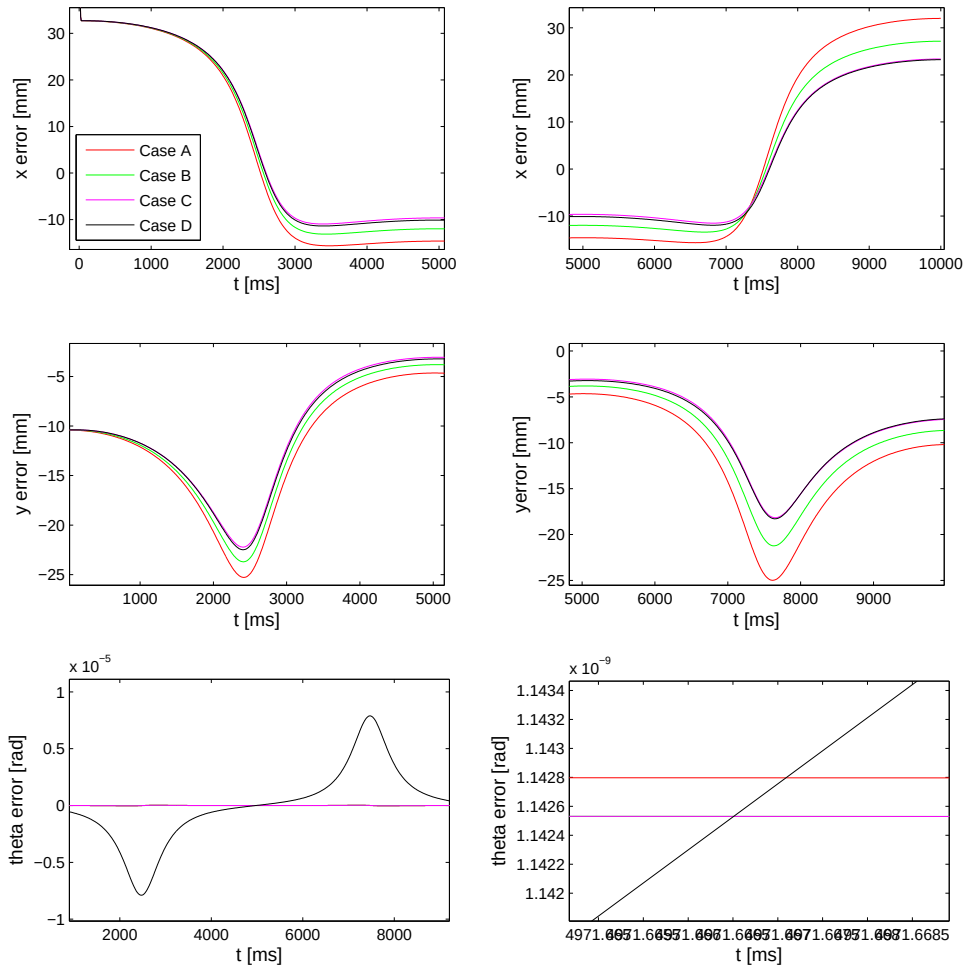

Figure 17: Simulations with different values for Np and Nc: errors in x,y and θ.

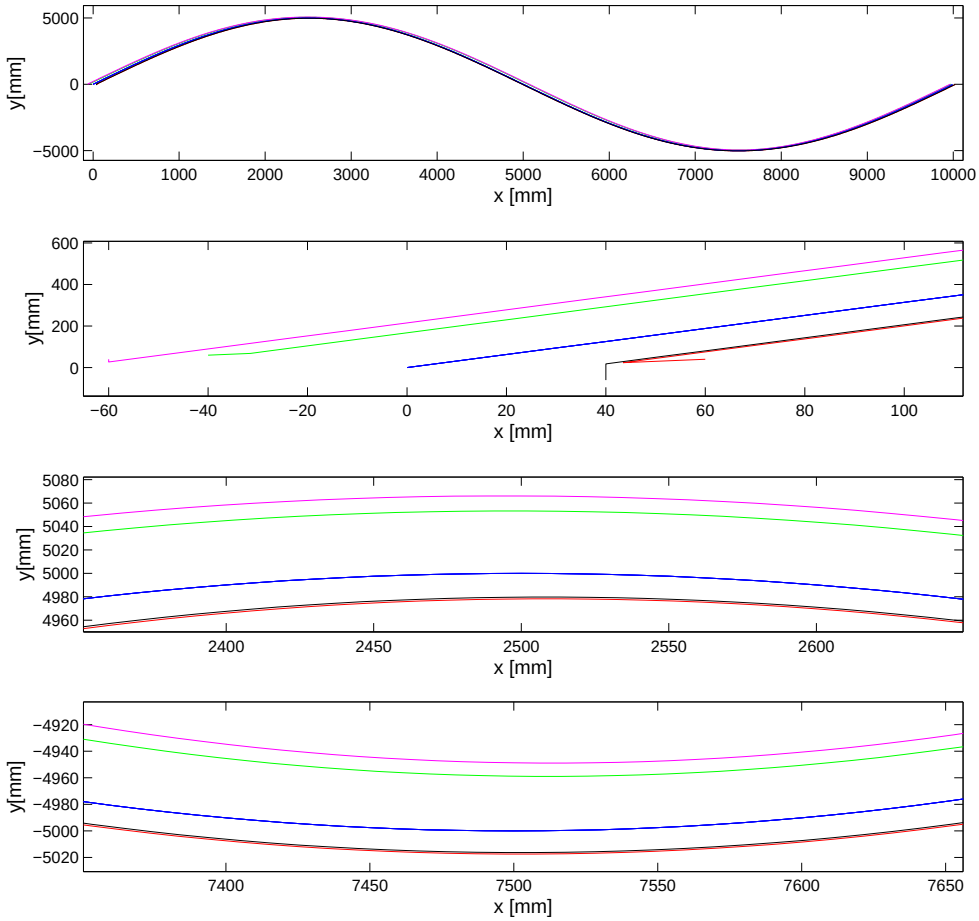

Figure 18: Simulations with different initial conditions with a sinusoidal reference trajectory: different regions in the x-y plane.

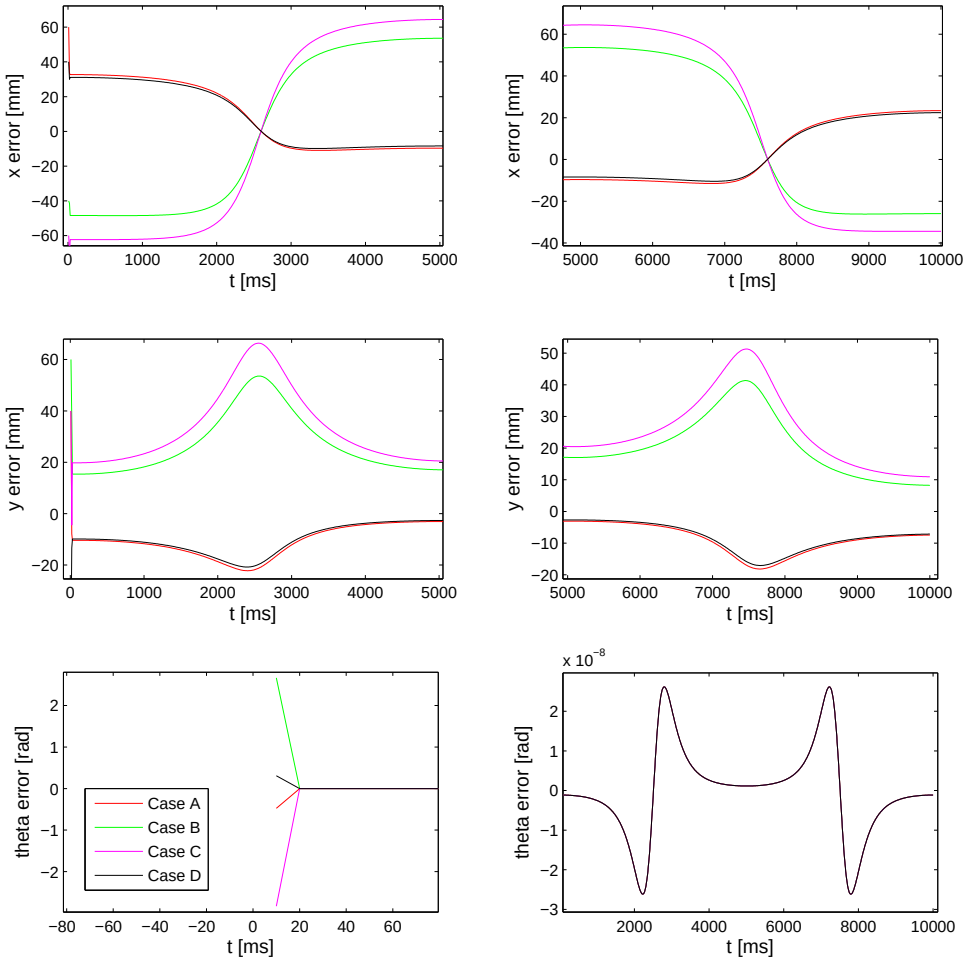

Figure 19: Simulations with different initial conditions with a sinusoidal reference trajectory: errors in x,y and θ.

6.4 Reference trajectory = x axis, with initial error

Constants	Symbol	Value
Reference trajectory		**x** axis
Prediction horizon	N_p	10
Control horizon	N_c	9
x coordinate error weight in matrix Q	Q_x	10000
y coordinate error weight in matrix Q	Q_y	10000
θ coordinate error weight in matrix Q	Q_θ	10000
CASE **A**: Variables	Symbol	Value
Initial state vector ([mm],[mm],[rad])	$\mathbf{x}_0$	(60, 40, $\pi/4$)
CASE **B**: Variables	Symbol	Value
Initial state vector ([mm],[mm],[rad])	$\mathbf{x}_0$	(-40, 60, $5\pi/4$)
CASE **C**: Variables	Symbol	Value
Initial state vector ([mm],[mm],[rad])	$\mathbf{x}_0$	(-60,- 40, $-\pi/2$)
CASE **D**: Variables	Symbol	Value
Initial state vector ([mm],[mm],[rad])	$\mathbf{x}_0$	(40, -60, $\pi/2$)

Table 5: Table for reference trajectory = **x** axis, with different initial errors.

Figure 21 shows the errors after the first sampling period. We can see that the errors in the three components of the state remain constant from the instant $t = 30ms$, that is after the third sampling period.

In the case of components x and θ, the error becomes zero at a certain instant, $t = 30ms$ and $t = 20ms$ respectively. However, the error in y remains constant once the angle tracking is good. Therefore, an offset in y remains during the whole simulation.

Case A is the one that better follows the reference, because the initial error in θ is only $\pi/4$, that is the smallest of all the cases tried.

In case B, since the initial error in θ is quite big, $-3\pi/4$, the system starts going in the wrong direction, and it gets further from the trajectory, making the error in x become bigger and the error in y smaller.

In cases C and D, the initial errors in θ are $-\pi/2$ and $\pi/2$ respectively. Then the system starts going further from the reference, but not as fast as in case B, because the errors are smaller. In these cases, at the beginning, the error in x becomes slightly bigger because they do not increase the value of the x component with the time and the reference does this.

Conclusions:

- The magnitude of the errors, both in x and y during the whole simulation is highly related to the initial error in θ.
- Once again, the offset in y, does not change its sign.

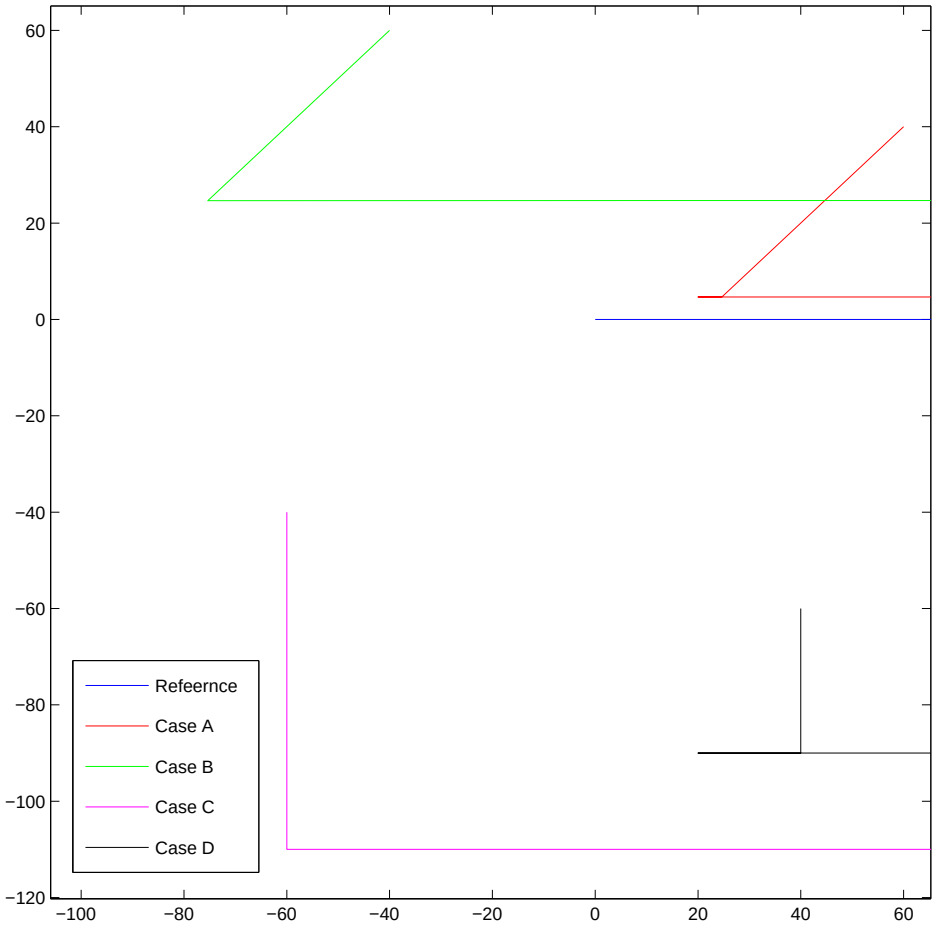

Figure 20: Simulations with different initial conditions with the **x** axis as reference trajectory: different regions in the x-y plane.

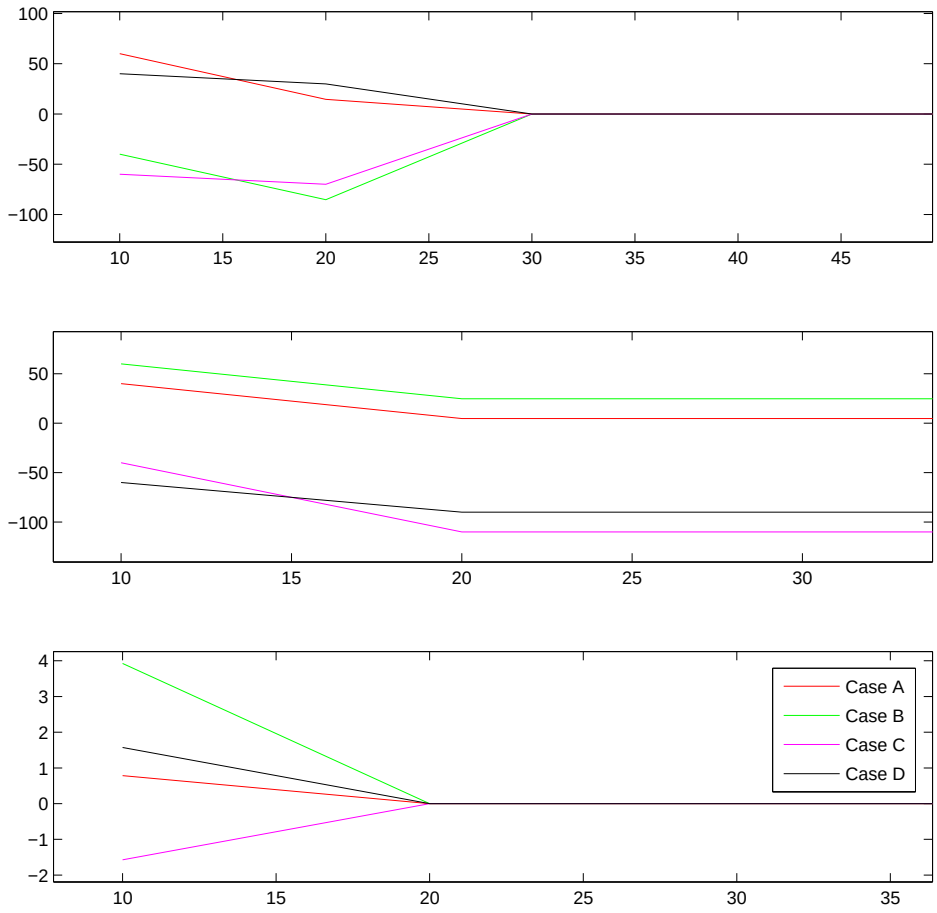

Figure 21: Simulations with different initial conditions with the **x** axis as reference trajectory: errors in x,y and θ.

- From the very beginning of the simulation, the errors remain constant and equal to zero in the case of x and θ; and constant, but not zero, in the case of y.
- The error in y remains constant once the error in θ is zero. However the error in x, can keep on becoming smaller, after this instant, until it reaches zero.

6.5 Reference trajectory = y axis, with initial error

Constants	Symbol	Value
Reference trajectory		**y** axis
Prediction horizon	N_p	6
Control horizon	N_c	5
x coordinate error weight in matrix Q	Q_x	100
y coordinate error weight in matrix Q	Q_y	100
θ coordinate error weight in matrix Q	Q_θ	100
CASE **A**: Variables	Symbol	Value
Initial state vector ([mm],[mm],[rad])	$\mathbf{x}_0$	(60, 40, $\pi/4$)
CASE **B**: Variables	Symbol	Value
Initial state vector ([mm],[mm],[rad])	$\mathbf{x}_0$	(60, 40, $\pi/4$)
CASE **C**: Variables	Symbol	Value
Initial state vector ([mm],[mm],[rad])	$\mathbf{x}_0$	(60, 40, $\pi/4$)
CASE **D**: Variables	Symbol	Value
Initial state vector ([mm],[mm],[rad])	$\mathbf{x}_0$	(60, 40, $\pi/4$)

Table 6: Table for reference trajectory = **y** axis, with different initial errors.

Figure 23 shows the errors after the first sampling period. We can see that the errors in the three components of the state remain constant from the instant $t = 30ms$, that is after the third sampling period.

In the case of components y and θ, the error becomes zero at a certain instant, $t = 30ms$ and $t = 20ms$ respectively. However, the error in x remains constant once the angle tracking is good. Therefore, an offset in x remains during the whole simulation. Then, this is the analogous case as the previous one, when the reference trajectory was the x axis, but exchanging the behaviors of x and y.

Cases D and C, have initial errors in θ equal to 0 and π respectively, it makes that the errors in x remain constant and equal to the initial ones for the whole simulation. However, in cases A and B, they become smaller.

The errors in y increase at the beginning in cases B and C, which are the cases which higher errors in θ, $3\pi/2$ and π respectively. In the other cases, the error in y decreases.

Conclusions:

- The magnitude of the errors, both in x and y during the whole simulation is highly related to the initial error in θ.
- Once again, the offset in y, does not change its sign.
- From the very beginning of the simulation, the errors remain constant and equal to zero in the case of y and θ; and constant, but not zero, in the case of x.

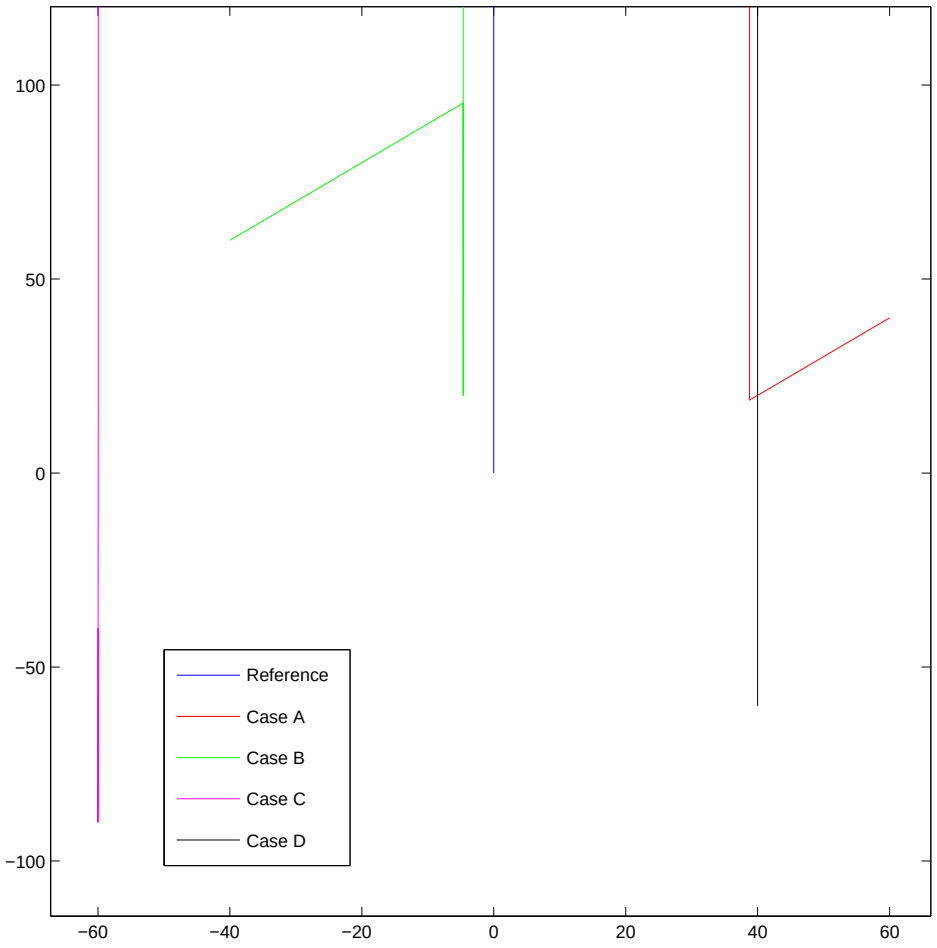

Figure 22: Simulations with different initial conditions with the **y** axis as reference trajectory: different regions in the x-y plane.

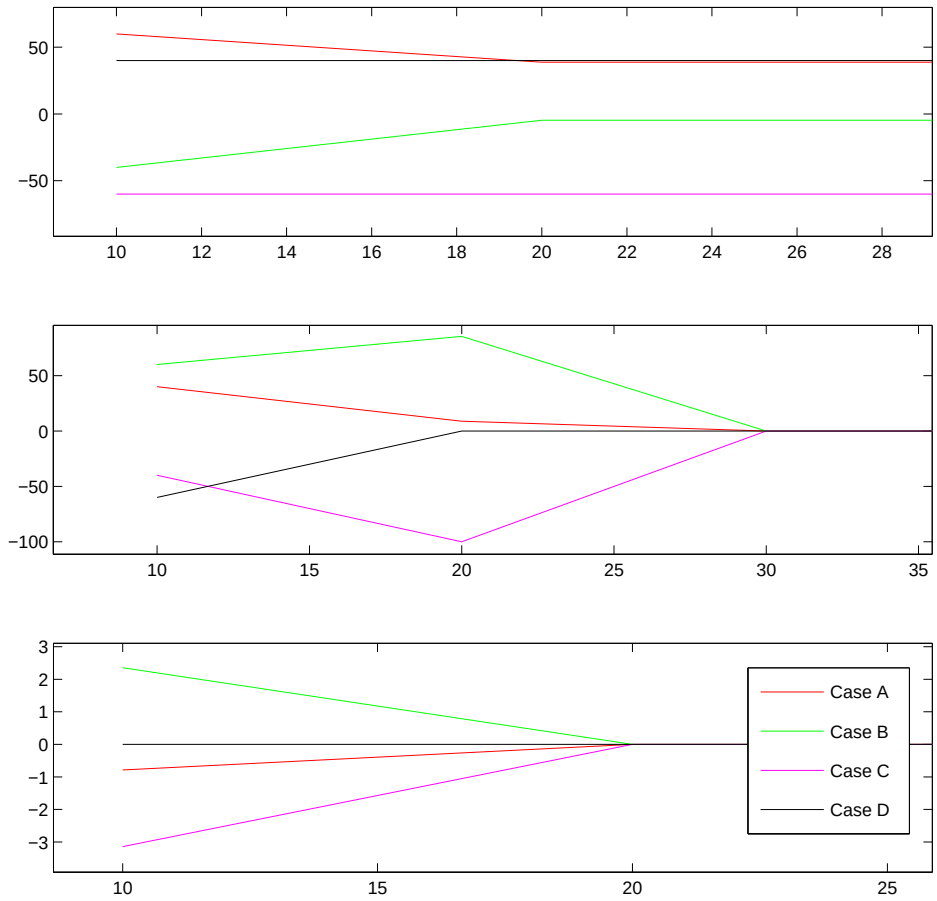

Figure 23: Simulations with different initial conditions with the **y** axis as reference trajectory: errors in x,y and θ.

- The error in x remains constant once the error in θ is zero. However the error in y, can keep on becoming smaller, after this instant, until it reaches zero.

7 EBCUF simulations in Matlab

In section 4, the theory of the Estimator Based Control Using Flatness was explained. Then, in section 4.4, different variations for the EBCUF algorithm were presented.

The previously presented MPC algorithm was implemented in Matlab in **discrete time**, while the EBCUF algorithm is implemented using the differential equations of the system, i.e., **continuous time**. Particularly, using *Simulik*, a block diagram has been created and then, every 'S-Function' block used has been defined with an independent matlab file. Finally, the whole simulation is called from another matlab file.

For the simulations in this section, the variable-step solver *ode45 (Dormand-Price)* has been used. However, before the experimentation part, other simulations where done using the variable-step solver *ode3 (Bogacki-Sharpine)*, in order to simulate the sampling period that it would be possible to have in practice. This sampling period is basically imposed by the speed of the camera to take the measurements, the image treatment to get the current state, the filtering of these measures, the solution of the control algorithm and the communication with the robot to send the inputs to the motors, as it is shown in section 9.

In this section, the results obtained are presented. Not all of them show the same level of performance, however it is interesting to see the different options.

- **Reference trajectory:**

In this section, 7, the reference trajectory is **not** explicitly determined. The **only** reference is a point, given by x_{ref} and y_{ref}, and the speeds in x and y, given by $\dot{x}_{ref}$ and $\dot{y}_{ref}$.

The **reference** used is $[x_{ref}, y_{ref}, \dot{x}_{ref}, \dot{y}_{ref}] = [0, 0, 0, 0]$, which stays constant for the whole simulation section.

Even if the trajectory is **not** explicitly defined, there is a way to somehow control it. The way to do it is by using model (38) instead of model (36). When doing so, we take advantage of the fact that the model is **decoupled** and we consider the two chains of integrators corresponding to the control of x and y respectively.

Then, different matrices L and K can be obtained for the estimator and controller of each integrators chain. Hence, when doing the pole placement of $(\mathbf{A} - \mathbf{LC})$ and $(\mathbf{A} - \mathbf{BK})$, different poles can be chosen for each chain. Therefore it is possible to get a faster or slower control of one of the components with respect to the other. Also by tuning the matrices L and K of each chain, the **final orientation** can be qualitatively controlled.

For instance, choosing the poles to find L and K to control y with higher absolute values than those to find L and K to control x, the error in y decreases faster than the one in x. Hence it makes the final orientation to be closer to the x axis rather than the y one, i.e., closer to 0 or π than to $\pi/2$ or $3\pi/2$.

Notation:
$\mathbf{K_1}$ = matrix for the controller (47) in the x component chain of integrators, found by pole placement of $(\mathbf{A_1} - \mathbf{B_1K})$ (39).
$\mathbf{K_2}$ = matrix for the controller (47) in the y component chain of integrators, found by pole placement of $(\mathbf{A_2} - \mathbf{B_2K})$ (39).
$\mathbf{L_1}$ = matrix for the estimator (41) in the x component chain of integrators, found by pole placement of $(\mathbf{A_1} - \mathbf{LC_1})$ (39).
$\mathbf{L_2}$ = matrix for the estimator (41) in the y component chain of integrators, found by pole placement of $(\mathbf{A_2} - \mathbf{LC_2})$ (39).

To simplify, the poles chosen for K_i and L_i with $i = 1, 2$ are the same. Then we have that:
$x - poles$ = poles to place K_1 and L_1 $y - poles$ = poles to place K_2 and L_2

7.1 Case A:

As explained in section 4.4.1, this controller does not include θ in the closed loop, because the measurements are only x and y. A way to do a kind of open loop control on the orientation is to give the initial angle to the estimator through the initial values of $\hat{x}$ and $\hat{y}$.

Both v and ω are calculated by using flatness, i.e., through equations (34a) and (34b) respectively.

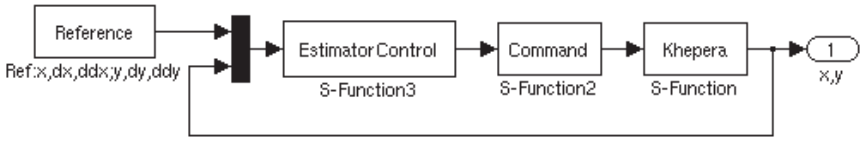

Figure 24: Block diagram for case A

The common parameters for these simulations have been the following ones:

Note: As in section 6, the **length** and **time units** do not affect the results obtained in simulation. Hence, the units here stated are just used as a tool to give the results in a more clear way. However, for the experimentation, it is necessary to adjust the needed units that imply a **feasible command** and a **possible control** of the real system, taking also into account the **state measurements acquisition speed**.

- **Case A1:**

  In this case, the estimator initial conditions do not consider the initial orientation. Therefore, since θ is not controlled in any way, the system does not converge to the origin as it was supposed to do.

Parameters	Symbol	Value
Reference trajectory	$[x_{ref}, y_{ref}, \dot{x}_{ref}, \dot{y}_{ref}]$	[0, 0, 0, 0]
State initial conditions ([cm],[cm],[rad])	$\mathbf{x}_0$	$[-2, -1, 0]$
CASE **A1**:	Symbol	Value
Estimator initial conditions	$[\hat{x}, \hat{\dot{x}}, \hat{y}, \hat{\dot{y}}]$	$[0, 0, 0, 0]$
x-Control	$x - poles$	$[-1, -1]$
y-Control	$y - poles$	$[-1.4, -1.4]$
CASE **A2**:	Symbol	Value
Estimator initial conditions	$[\hat{x}, \hat{\dot{x}}, \hat{y}, \hat{\dot{y}}]$	$[0, 1, 0, 0]$
x-Control	$x - poles$	$[-1, -1]$
y-Control	$y - poles$	$[-1.4, -1.4]$
CASE **A3**:	Symbol	Value
Estimator initial conditions	$[\hat{x}, \hat{\dot{x}}, \hat{y}, \hat{\dot{y}}]$	$[0, 1, 0, 0]$
x-Control	$x - poles$	$[-2, -2]$
y-Control	$y - poles$	$[-4, -4]$

Table 7: Table with the parameters for different simulations of case A.

- **Case A2:** In this case, introducing the orientation with the estimator initial conditions, the open loop in θ is archived and therefore the system reaches the origin.

  However, even when introducing the initial orientation through the initial state of the estimator, the system is very sensible to the choice of poles. The poles chosen for this case, are the best that have been found varying them for the x and y control, but keeping them constant to find K_i and L_i for each chain.

- **Case A3:**

  In this case, even if taking into account the initial orientation, the choice of poles makes the system diverge from the reference point.

7.2 Case B

In this case, the measures of the whole state are available. Then, v is calculated by applying flatness (34a) and ω, using a proportional controller (K_p). That is, using equations

$$\hat{\theta} = \arctan \hat{\dot{y}}/\hat{\dot{x}} \tag{72}$$

$$\omega = K_p(\hat{\theta} - \theta) \tag{73}$$

(It has to be taken into account that the function arctan has two solutions and that the function arctan 2 of Matlab has to be used in order to get a unique solution from $-\pi$ to π).

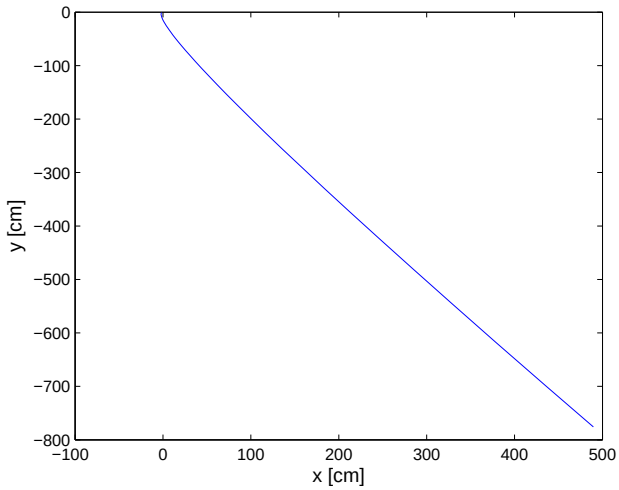

Figure 25: Trajectory tracking in case A1, without considering the initial orientation through the initial conditions of the estimator.

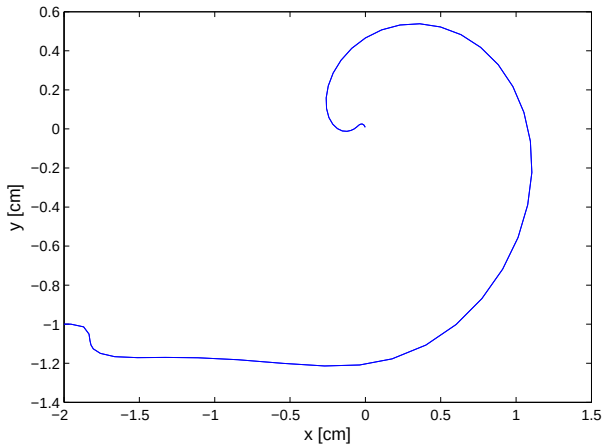

Figure 26: Trajectory tracking in case A2, considering the initial orientation through the initial conditions of the estimator and $x - poles = [-1, -1]$ and $y - poles = [-1.4, -1.4]$.

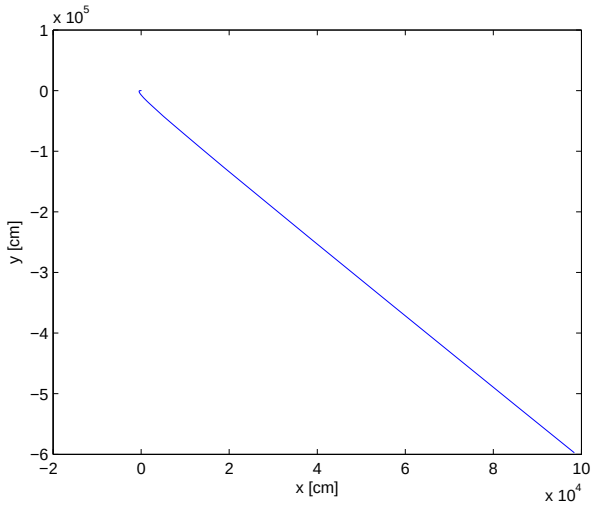

Figure 27: Trajectory tracking in case A3, considering the initial orientation through the initial conditions of the estimator and $x - poles = [-2, -2]$ and $y - poles = [-4, -4]$.

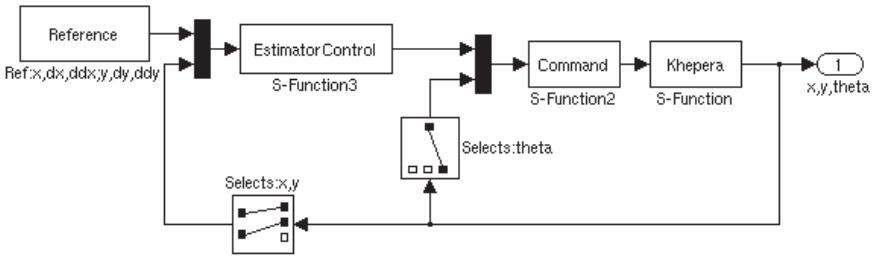

Figure 28: block diagram for cases B and C

Parameters	Symbol	Value
Reference trajectory	$[x_{ref}, y_{ref}, \dot{x}_{ref}, \dot{y}_{ref}]$	[0, 0, 0, 0]
State initial conditions ([cm],[cm],[rad])	$\mathbf{x}_0$	$[-2, -1, 0]$
Estimator initial conditions	$[\hat{x}, \hat{\dot{x}}, \hat{y}, \hat{\dot{y}}]$	$[0, 0, 0, 0]$
x-Control	$x - poles$	$[-2, -2]$
y-Control	$y - poles$	$[-4, -4]$
$Proportionalcontroller$	K_p	20

Table 8: Table with the parameters of case B.

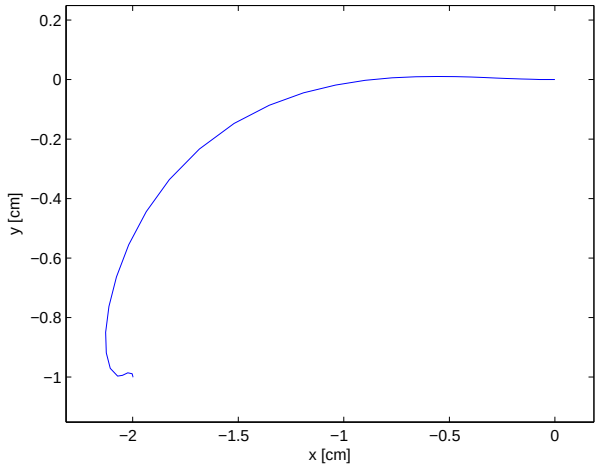

Figure 29: Trajectory tracking in case B, which uses a proportional controller for the angular speed ω.

7.3 Case C

This case calculates the input v using flatness (34a) and ω using flatness and a proportional controller as in equation (69) here restated

$$\omega = \frac{-\hat{\dot{x}}\hat{y} + \hat{\dot{x}}\hat{\dot{y}}}{\hat{\dot{x}}^2 + \hat{\dot{y}}^2} + K_p(\hat{\theta} - \theta) \tag{74}$$

The block diagram and the parameters for this case are the same as in case B. And as well as in case B, the system reaches the reference point.

7.4 Effect of parameter variations

Now the different cases have been presented, some simulations to see the effect of some parameters have been done. All the simulations in this subsection have been

Parameters	Symbol	Value
Reference trajectory	$[x_{ref}, y_{ref}, \dot{x}_{ref}, \dot{y}_{ref}]$	[0, 0, 0, 0]
State initial conditions [[cm],[cm],[rad]]	$\mathbf{x}_0$	$[-2, -1, 0]$
Estimator initial conditions	$[\hat{x}, \hat{\dot{x}}, \hat{y}, \hat{\dot{y}}]$	$[0, 0, 0, 0]$
x-Control	$x - poles$	$[-2, -2]$
y-Control	$y - poles$	$[-4, -4]$
Proportional controller	K_p	20

Table 9: Table with the parameters of case C.

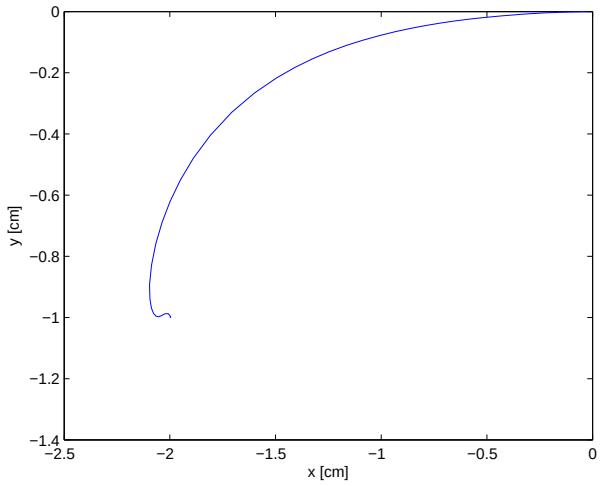

Figure 30: Trajectory tracking in case C, which uses flatness and a proportional controller for the angular speed ω.

done using **case C**, with the simulation parameters used before except the ones that are explicitly restated.

Parameters	Symbol	Value
Reference trajectory	$[x_{ref}, y_{ref}, \dot{x}_{ref}, \dot{y}_{ref}]$	[0, 0, 0, 0]
State initial conditions ([cm],[cm],[rad])	$\mathbf{x}_0$	$[-2, -2, 0]$
Estimator initial conditions	$[\hat{x}, \hat{\dot{x}}, \hat{y}, \hat{\dot{y}}]$	$[0, 0, 0, 0]$

Table 10: Table with the parameters common to all simulations to study the effect of the poles and K_p.

Notation:
$K_x - poles$ = control poles for the x component

$K_y - poles$ = control poles for the y component
$L_x - poles$ = estimation poles for the x component
$L_y - poles$ = estimation poles for the y component

7.4.1 Effect of the magnitude of the $x, y - poles$

For these simulations, $K_x - poles = K_y - poles = L_x - poles = L_y - poles$; $K_p = constant$.

Case	$K_x - poles$	$K_y - poles$	$L_x - poles$	$L_y - poles$	K_p
Case A	$[-0.5, -0.5]$	$[-0.5, -0.5]$	$[-0.5, -0.5]$	$[-0.5, -0.5]$	20
Case B	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	20
Case C	$[-10, -10]$	$[-10, -10]$	$[-10, -10]$	$[-10, -10]$	20
Case D	$[-30, -30]$	$[-30, -30]$	$[-30, -30]$	$[-30, -30]$	20

Table 11: Poles chosen to study the effect of their magnitude.

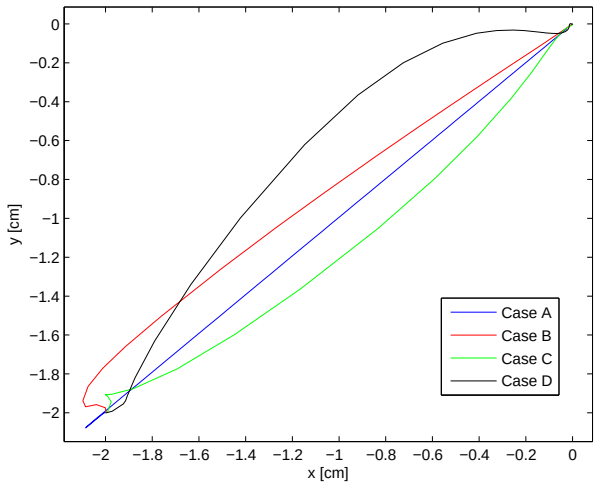

Figure 31: Trajectories tracking with different magnitudes of the $x, y - poles$. In all cases, the origin is reached.

Figure 31 shows how the four cases reach the origin without any problem. Figure 32 is a part of 31 and shows that the higher the poles are, the more oscillations there are next to the final point. These oscillations are produced by the fact that greater poles make the controller have a faster dynamics which have to be also followed by the orientation, θ. Therefore, to archive that, the proportional gain K_p has also to increase if the poles are faster.

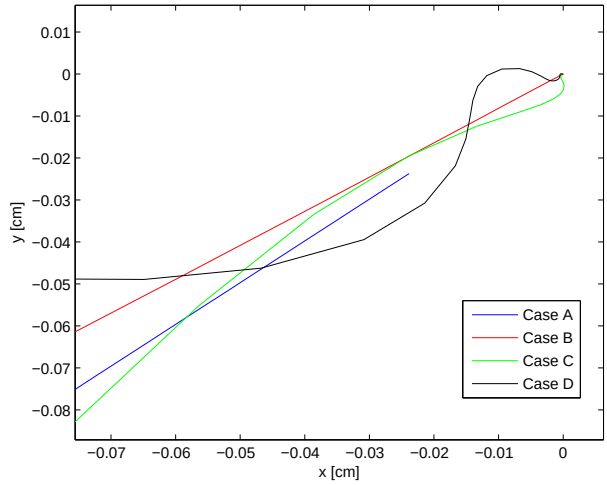

Figure 32: Region close to the reference point, i.e., final part of the simulation.

In figure 33 we can observe that the higher the poles are, the faster the system is. Both x and y coordinates reach the reference much faster. For case A, since the poles are too low, after $15s$ of simulation the system has not exactly arrived to the origin yet. We can also see it in figure 32.

Then, figure 34 effectively shows that the speeds, both linear and angular, increase with the poles magnitude. Also in this figure, we can see that increasing too much the poles, makes the angular speed input oscillate.

7.4.2 Effect of the magnitude of K_p

For these simulations, $L_x - poles = L_y - poles = K_x - poles = K_y - poles = constant$; $K_p = variable$.

Case	$K_x - poles$	$K_y - poles$	$L_x - poles$	$L_y - poles$	K_p
Case A	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	5
Case B	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	10
Case C	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	20
Case D	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	40

Table 12: Poles chosen to study the effect of the magnitude of K_p.

Looking at figure 35 we confirm that the higher the magnitude of K_p with respect to the magnitude of the poles, the straighter the line between the initial and the final point is, which is something positive. The negative point is that, the higher the magnitude of K_p, the higher the overshooting in the orientation is; i.e., the system

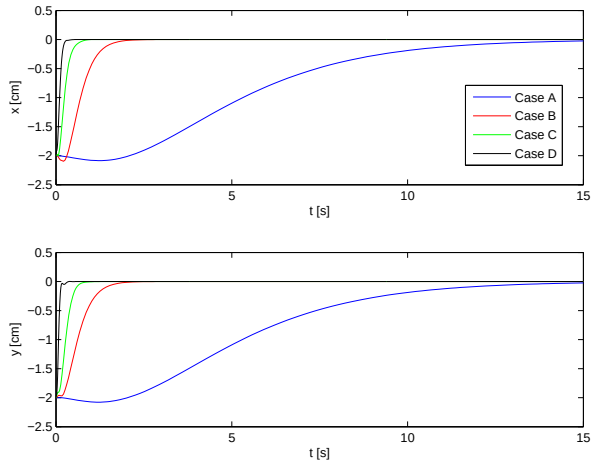

Figure 33: Evolution in time of the x and y coordinates, varying the magnitude of the $x, y - poles$.

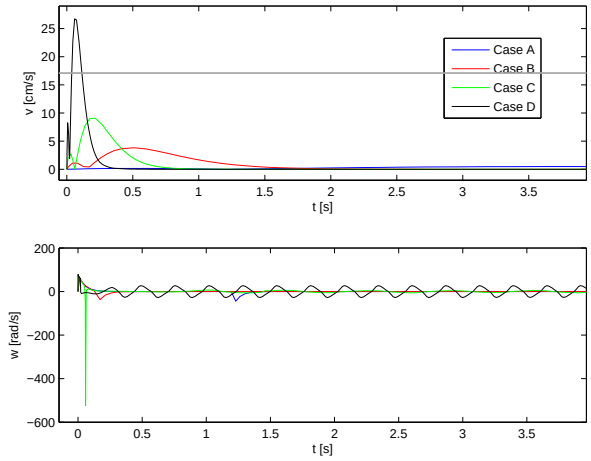

Figure 34: Evolution in time of the inputs (linear and angular speeds) varying the magnitude of the $x, y - poles$

tries to correct faster the initial orientation error. However, in cases B, C and D, we can observe how this overshooting increases from one case to the other.

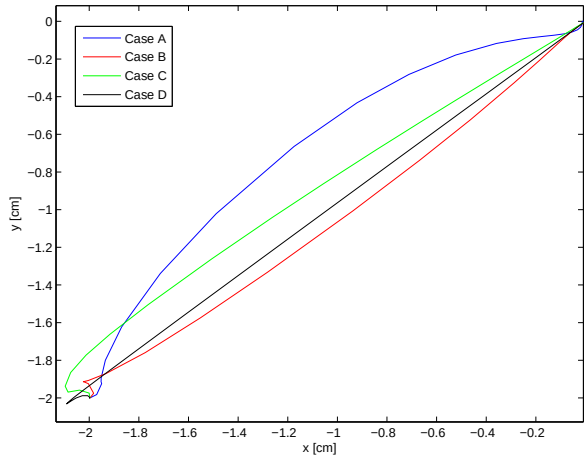

Figure 35: Trajectories tracking with different K_p.

In figure 36, we observe that the linear speeds are similar for the four cases. However, the initial angular speed increases with K_p.

7.4.3 Effect of different poles for x and y

Now, let us verify that the magnitude of the poles chosen determines the magnitude of the gain K_p needed to get good performance. For these simulations, $K_x - poles = L_x - poles$; $K_y - poles = L_y - poles$; $K_p = constant$.

Case	$K_x - poles$	$K_y - poles$	$L_x - poles$	$L_y - poles$	K_p
Case A	$[-1, -1]$	$[-4, -4]$	$[-1, -1]$	$[-4, -4]$	20
Case B	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	20
Case C	$[-4, -4]$	$[-1, -1]$	$[-4, -4]$	$[-1, -1]$	20

Table 13: Poles chosen to study the effect of a difference between the $x - poles$ and the $y - poles$.

Figure 37 shows how the trajectory varies when choosing different $x - poles$ and $y - poles$. The higher the magnitude of the poles in one component, the faster the error in this component decreases.

We can see that case A has slower $x - poles$, that is why the system starts decreasing the error in y and then in x. It makes reach the origin with an angle close

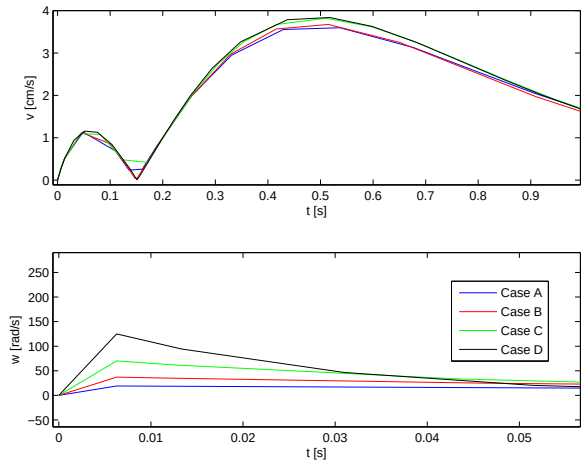

Figure 36: First instants, where the difference in K_p has a higher effect in the evolution in time of the inputs v and ω.

to the y-axis. In this case it is close to zero, but if the initial x coordinate had been positive it would have made the final orientation close to π.

Case B, reduces the error at the same speed for both coordinates because the poles are the same for the control of each one.

Finally, case C starts reducing the error in x and then in y. It makes the final orientation be close to $\pi/2$; if the initial y coordinated had been positive, the final orientation would have been close to $-\pi/2$.

Hence, we see that making the $x-poles$ and $y-poles$ different we can qualitatively control the final orientation and the trajectory tracking.

Figure 38 shows the evolution of the x and y coordinates, and also the error in them, since the reference point is the origin. We can effectively see that the higher the poles in a component, the faster the error is reduced. That is, the order from faster to slower cases for the x coordinate is: C = B, A; and for the y coordinate is: A = B,C.

7.4.4 Effect of the magnitude of L_i

For these simulations, $K_x - poles = K_y - poles = constant$; $L_x - poles = L_y - poles = variable$; $K_p = constant$.

We can observe in figure 39 that the faster the $L_i - poles$ are, the faster the estimator converges to the real position. It is clear for the x and y coordinates, however for θ the behavior tendency is not that clear.

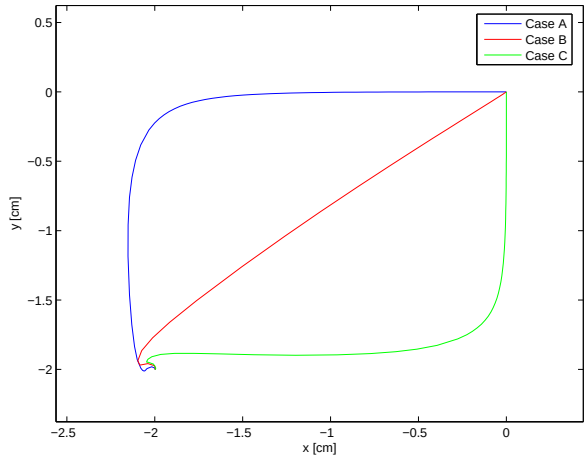

Figure 37: Trajectories tracking with a difference between the $x - poles$ and the $y - poles$.

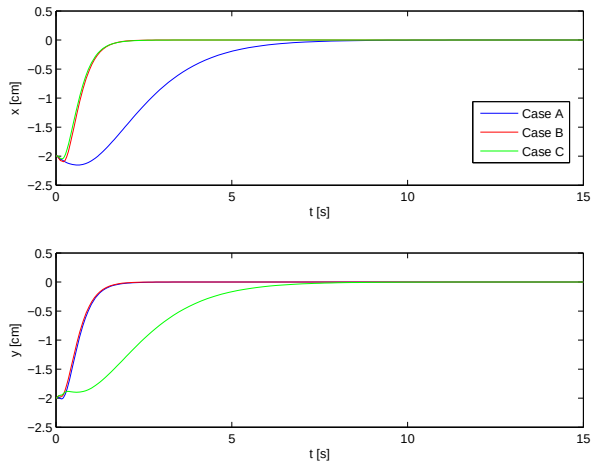

Figure 38: Evolution in time of the x and y coordinates with a difference between the $x - poles$ and the $y - poles$.

Case	$K_x - poles$	$K_y - poles$	$L_x - poles$	$L_y - poles$	K_p
Case A	$[-4, -4]$	$[-4, -4]$	$[-1, -1]$	$[-1, -1]$	20
Case B	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	20
Case C	$[-4, -4]$	$[-4, -4]$	$[-8, -8]$	$[-8, -8]$	20

Table 14: Poles chosen to study the effect of the magnitude of the $L_i - poles$.

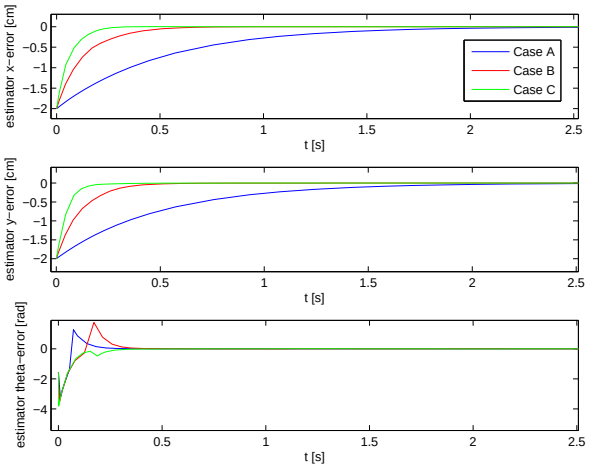

Figure 39: Error between the real state coordinates and the estimated ones with different $L_i - poles$.

7.4.5 Effect of different poles for K_i

For these simulations, $L_x - poles = L_y - poles = constant$; $K_x - poles; = K_y - poles = variable$; $K_p = constant$.

Case	$K_x - poles$	$K_y - poles$	$L_x - poles$	$L_y - poles$	K_p
Case A	$[-1, -1]$	$[-1, -1]$	$[-4, -4]$	$[-4, -4]$	20
Case B	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	20
Case C	$[-8, -8]$	$[-8, -8]$	$[-4, -4]$	$[-4, -4]$	20

Table 15: Poles chosen to study the effect of the magnitude of the $K_i - poles$.

Figure 40 shows the errors of the controller for different values of the $K_i - poles$. These errors are the differences between the reference state components and the estimated state components. We can see how the higher the $K_i - poles$ are, the faster these errors become zero. That is why the system real state also converges to the reference faster, as we can see in figures 41 and 42.

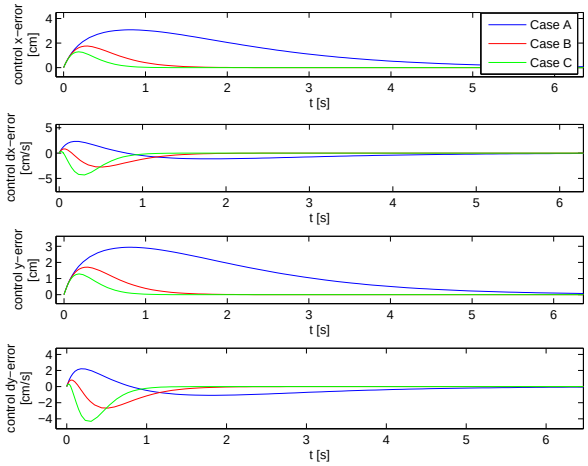

Figure 40: Evolution in time of the controller errors, i.e.,the difference between the reference state and the observed one, with different $K_i - poles$.

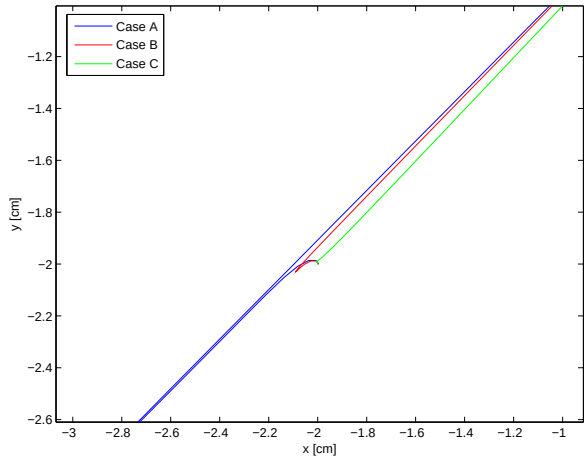

Figure 41: Initial part of the trajectories tracking with different $K_i - poles$.

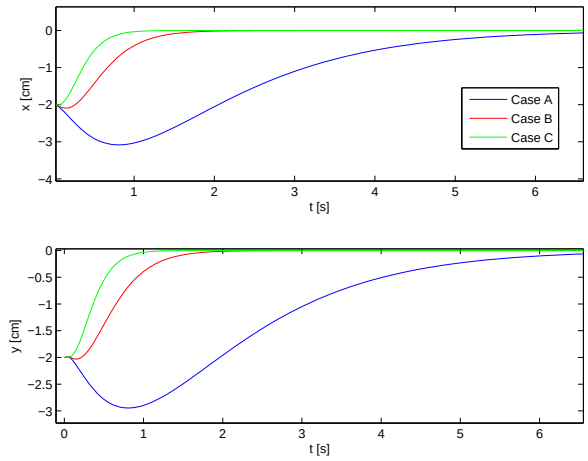

Figure 42: Evolution in time of the x and y coordinates with different $K_i - poles$.

8 MPC experimentation

For the experimental part of this project, LabVIEW has been used. Using this software, the following parts have been implemented:

1. Image acquisition obtained with a camera
2. Image treatment for the position estimation
3. Filtering of the image
4. Trajectory generation
5. Control algorithms
6. Communication with the robot

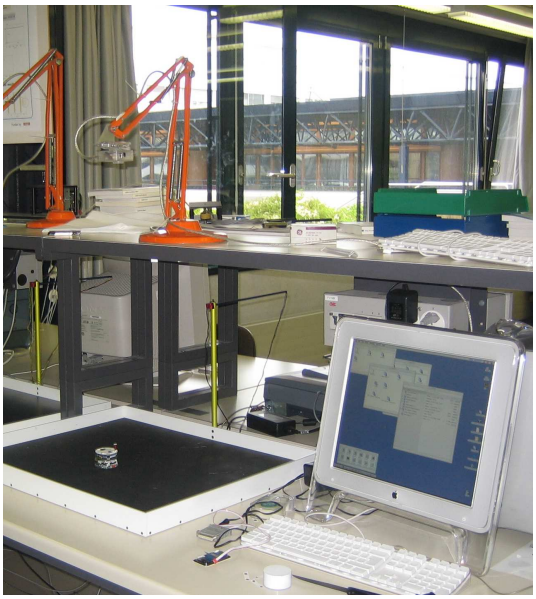

Figure 43: Experimental setup consisting on a camera, black surface, black and white robot and computer.

For the image acquisition obtained by the camera, a toolbox made by Christoph Saltzmann was used. The image treatment is based on the algorithms proposed by [Herrera, 1999] and the VIs (virtual instruments) developed by [J.M. Miamo ,2005]. The filtering of the image has also been developed by [J.M. Miamo ,2005]. And the control algorithms have been written by the author of this project.

The first MPC algorithm implemented was the one explained in section 3.4 and appendix B.4 using the reference angles, as [Niederberger, 2002] suggested. This approach was presented as the solution to the problem to calculate the matrix **F**.

The problem is that, as appendix B.4 presents, the matrix $\mathbf{F}$ is calculated from $\theta(k)$, $\theta(k\hat{+}1)$, ... , $\theta(k\hat{+}N_c)$, but at instant k, only $\theta(k)$ is known. Then the suggestion was to use the reference angles to be able to calculate $\mathbf{F}$.

This approach supposes the reference is known a priori. The results obtained with this algorithm were not good. The reason could be that calculating the matrix $\mathbf{F}$ using θ_{ref} is equivalent to approximate θ by θ_{ref} which could only be accepted for very small initial errors in the orientation.

If we consider (19), here restated

$$\hat{\mathbf{u}} = \mathbf{K}(\mathbf{r} - \hat{\mathbf{x}}^{\mathbf{0}}) \tag{75a}$$

where

$$\mathbf{K} = (\mathbf{F}^{\mathbf{T}}\mathbf{Q}\mathbf{F} + \mathbf{\Lambda})^{-\mathbf{1}}\mathbf{F}^{\mathbf{T}}\mathbf{Q} \in \Re^{m(N_c+1)\times nN_p} \tag{75b}$$

$$\hat{\mathbf{x}}^{\mathbf{0}} = \mathbf{Z}x(k) \in \Re^{nN_p} \tag{75c}$$

with the approach of [Niederberger, 2002], the error is actually introduced in $\mathbf{K}$. The matrix $\mathbf{Z}$ is not affected by this approximation because it is a known constant.

Searching for better results, another approach was taken. This one consists on using the measured angle $\theta_{meas}(k)$ and an **on line linear trajectory generation**. The current orientation, $\theta_{meas}(k)$, is used in the place of $\theta(k\hat{+}1)$, ... , $\theta(k\hat{+}N_c)$ for the calculation of $\mathbf{F}$. Doing that, an error is introduced once again in $\mathbf{K}$. Then, using an on line trajectory generation , r is also modified. The results applying this strategy are quite good. This report does not include any proof explaining the good behavior. However, we can intuitively see that the key point is the on line linear trajectory generation, which is explained as follows.

- **On line linear trajectory generation:**

The trajectory generation has been implemented in a VI which is used at each iteration. The **inputs** are:

- a final reference point
- a reference speed
- the sampling period
- the current robot position

With these inputs, the reference orientation that the robot should have to get to the final point is calculated. Then, the output is generated in two steps:

1. If the initial orientation is not the same as the calculated reference orientation (with a tolerance of $0.15rad$), the reference trajectory points (x_{ref}, y_{ref}) are

fixed to be the current position (x, y) . Doing this, the first thing to be corrected is the initial orientation error. Moreover, to make the turn smoother, the next rule was applied: if the orientation error was bigger than the tolerance, the turn was made step by step, by small increments $(0.01rad)$.

2. Once the real orientation reaches the reference angle (with the same tolerance as before), the trajectory is given by this constant reference angle and a sequence of equidistant (x, y) points, separated by the distance which is covered at the reference speed during a sampling period.

Using this trajectory generation and $\theta(k)$ for the calculation of $\mathbf{F}$, it was observed that the initial behavior of the robot was very sudden if the initial orientation error was big. Therefore, to improve this undesirable behavior, the trajectory generation was modified as follows:

- If the angle between the reference angle and the initial orientation, in absolute value, was smaller than $\pi/2$, the reference orientation was the one previously calculated. Therefore, after that, the MPC algorithm made the robot move forward (positive speeds for both wheels).
- On the other hand, if the angle between the reference angle and the initial orientation, in absolute value, was greater than $\pi/2$, the reference orientation previously calculated was corrected by adding πrad. Therefore, the MPC algorithm made the robot move backwards (negative speeds for both wheels).

Finally, a 5 mm 'ball' was added around the final reference point, to stop the motors of both wheels of the robot once it entered the ball. It was needed to do that, because the noise in the filtered values gave always an error, sometimes real and sometimes unreal, that made the controller permanently give non-zero inputs.

Doing this modification in the first step (reference orientation choice) of the trajectory generation, the results were quite good. That is why the experiments presented in this section have been done using this trajectory generation and the MPC control algorithm that includes the measures of $\theta(k)$ for the calculation of the matrix $\mathbf{F}$.

After the simulations done in Matlab, it was decided to carry on the following experiments: varying the weights in the matrix $\mathbf{Q}$ or varying N_p and N_c.

Note:

The measures obtained by the camera presented some noise which was reduced by using a Kalman filter. Also the fact of having a sampling period of $120ms$ (constant in all MPC experiments) which is quite big, has a filtering effect. However, even with this sampling period and the Kalman filter, some noise remained. That is why plotting the different curves in the same graph, as it was done in the simulations in Matlab, is in general not very useful because the noise makes the graph quite difficult to read. This is the reason why almost all the graphs are presented separately for every case in this section. It is clear that it makes it also difficult to compare and see the differences between cases, however, it has been thought to be the most appropriate way to present the results.

8.1 Effect of the matrix Q in experimentation

Subsection constants	Symbol	Value
Prediction horizon	N_p	6
Control horizon	N_c	5
CASE **A**: Subsection variables	Symbol	Value
x coordinate error weight in matrix Q	Q_x	100
y coordinate error weight in matrix Q	Q_y	100
θ coordinate error weight in matrix Q	Q_θ	100
CASE **B**: Subsection variables	Symbol	Value
x coordinate error weight in matrix Q	Q_x	10000
y coordinate error weight in matrix Q	Q_y	10000
θ coordinate error weight in matrix Q	Q_θ	10000
CASE **C**: Subsection variables	Symbol	Value
x coordinate error weight in matrix Q	Q_x	50
y coordinate error weight in matrix Q	Q_y	1000
θ coordinate error weight in matrix Q	Q_θ	10

Table 16: Table with the parameters for different structures of the matrix Q used in experimentation.

It can be observed in figure 44, how the trajectories followed in the three cases are very similar. Then, figures 45 and 46, confirm that the evolution in time of the x and y component are almost the same.

Figure 47 shows in the first subplot, the first instants of the orientation evolution for the three cases. We can say that case C is the one which has a more aggressive correction of the initial orientation error; followed by case B and finally case A. Looking at the other subplots in this figure, we can also see that in the evolution of the orientation case A presents more oscillations than the other cases and case C offers the smoothest curve.

Conclusions:

- Higher weights in the matrix **Q** make the initial orientation angle be corrected more aggressively and the evolution in time of the orientation be smoother.
- The x and y coordinates seem to be indirectly affected by **Q** because they depend on the orientation. However, a direct relationship has not been found.
- A proper tuning giving a different weight to each component seems to provide good results even if the magnitudes of the weights are not that high.
- The differences between cases are very small. It means the system is not very sensible to the variations of the weights in the matrix **Q**.

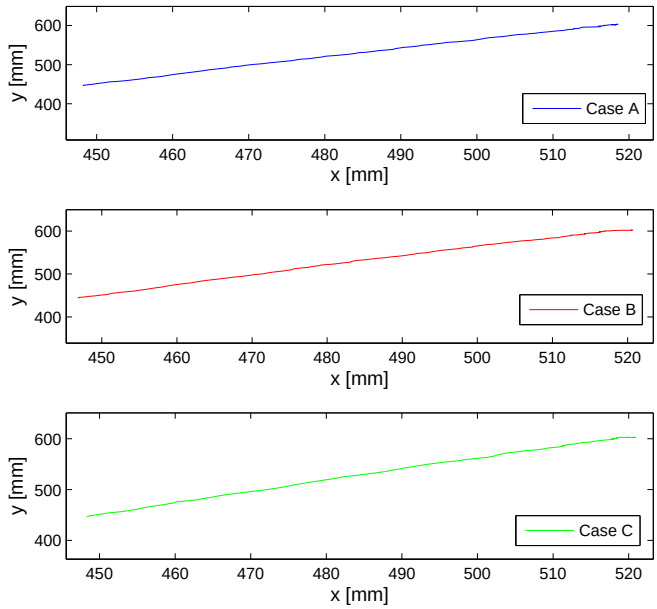

Figure 44: Trajectories followed in the different cases, varying the matrix **Q** weights.

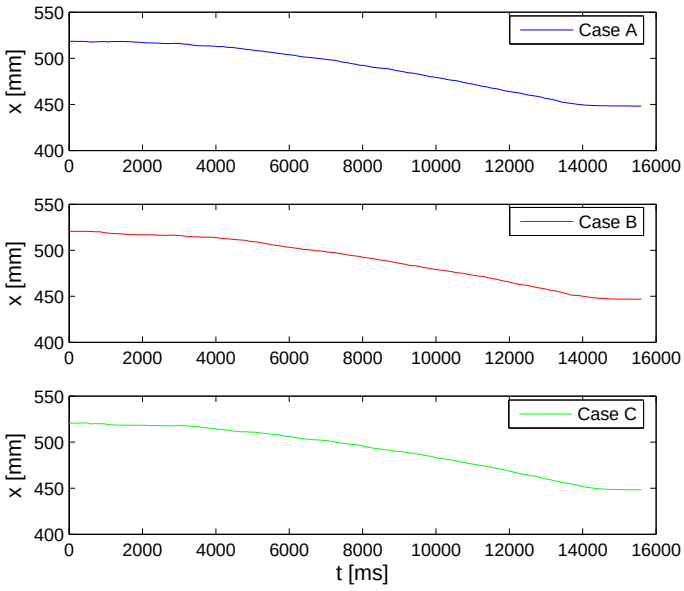

Figure 45: Evolution in time of the x component, for different **Q** weights.

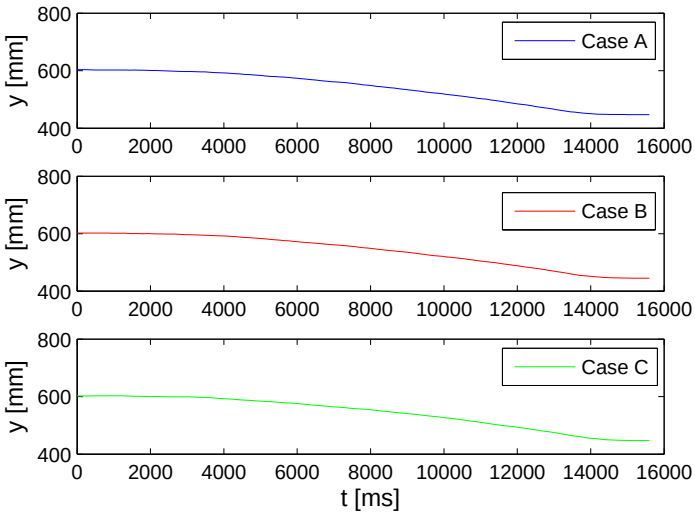

Figure 46: Evolution in time of the y component, for different $\mathbf{Q}$ weights.

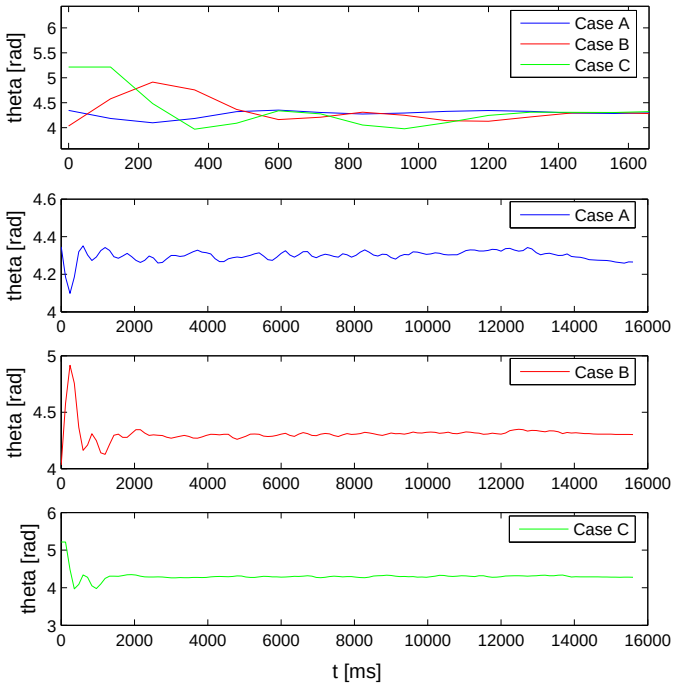

Figure 47: Evolution in time of the θ coordinate, for different $\mathbf{Q}$ weights.

8.2 Effect of N_p and N_c in experimentation

Subsection constants	Symbol	Value
x coordinate error weight in matrix Q	Q_x	100
y coordinate error weight in matrix Q	Q_y	100
θ coordinate error weight in matrix Q	Q_θ	100
CASE **A**: Subsection variables	Symbol	Value
Prediction horizon	N_p	2
Control horizon	N_c	1
CASE **B**: Subsection variables	Symbol	Value
Prediction horizon	N_p	6
Control horizon	N_c	5
CASE **C**: Subsection variables	Symbol	Value
Prediction horizon	N_p	10
Control horizon	N_c	9
CASE **D**: Subsection variables	Symbol	Value
Prediction horizon	N_p	10
Control horizon	N_c	1

Table 17: Table with the parameters for different prediction and control horizons: N_p andN_c.

Figure 48, presents the trajectories followed in the different cases, and it can be said once again, that in the four cases they are very similar. Figures 49 and 50, show separately the evolution of each component, x and y, and confirm they are almost the same in all cases.

Figure 51 shows the first instants of the orientation evolution for the four cases. It seems the order of cases from smaller to bigger oscillations is: A, B, C and D. However, the differences are really small and they could also be affected by the initial orientation of the robot which in practice can not be fixed with a 2 decimal precision, due to the filtered measures noise.

Finally, in figure 52, the first instants of the orientation evolution are presented separately. And it seems the case that presents more oscillations is case D, which corresponds to a higher difference between the values of N_p and N_c ($N_p = 10$ and $N_c = 1$).

Conclusions:

- Only a slightly more oscillatory behavior is observed in the evolution of the orientation when the difference between N_p and N_c becomes bigger.
- In general, no big differences between the cases are observed.

Final note after the MPCexperiments:

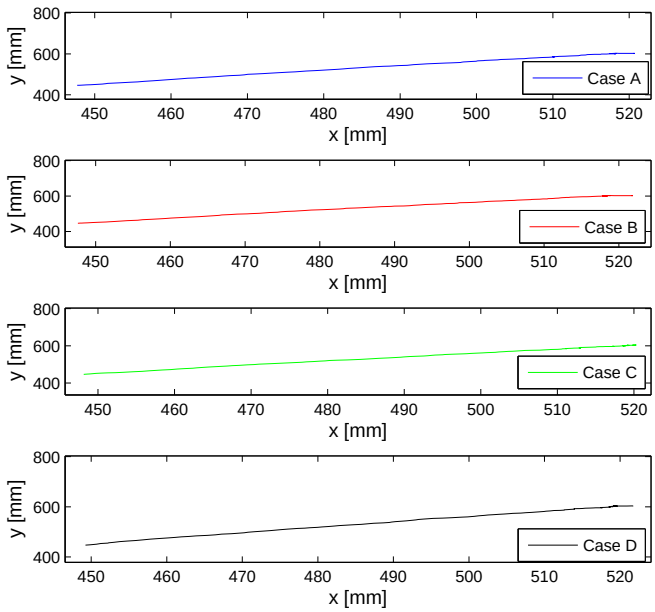

Figure 48: Trajectories followed in the different cases, varying N_p and N_c.

The differences observed between the different cases here presented are really small. It is probably due to the noise present even in the filtered values that have been used and also to the difficulties this noise presents when it comes to compare curves.

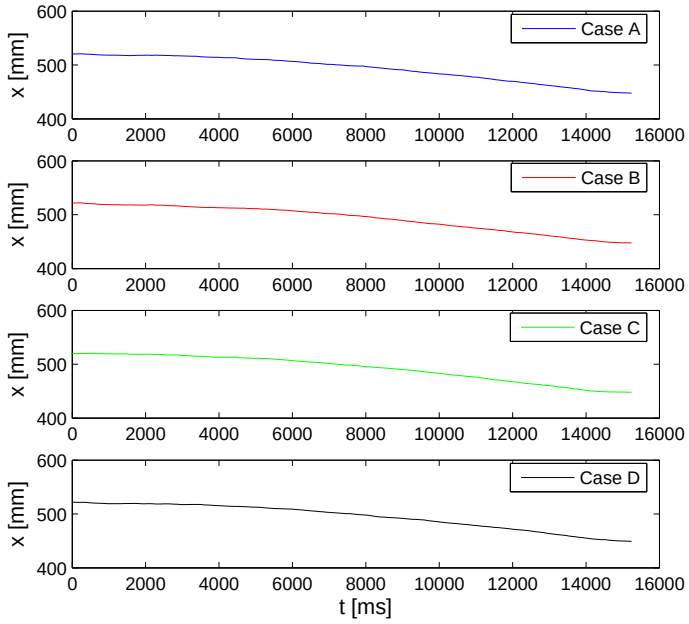

Figure 49: Evolution in time of the x component, for different values of N_p and N_c.

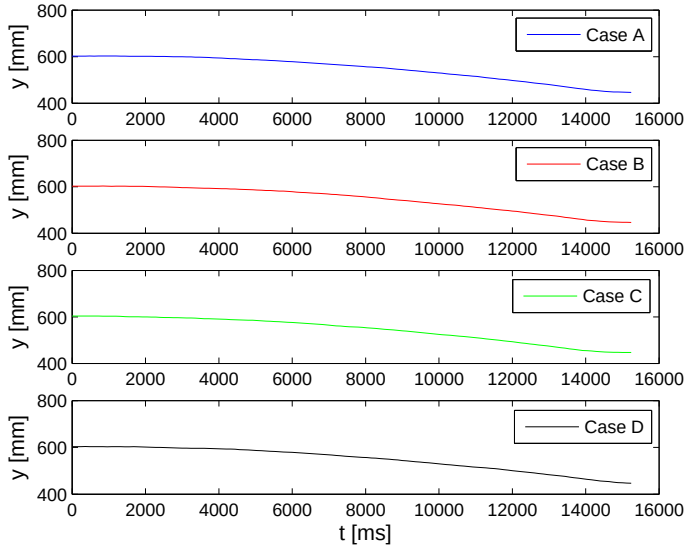

Figure 50: Evolution in time of the y component, for different values of N_p and N_c.

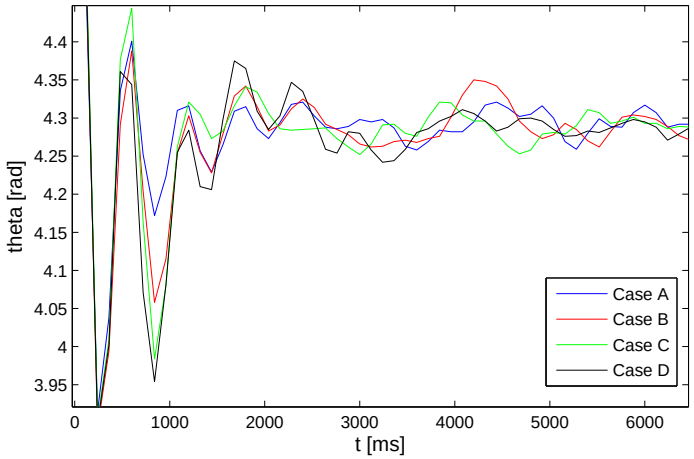

Figure 51: First instants of the evolution in time of the θ coordinate, for different values of N_p and N_c.

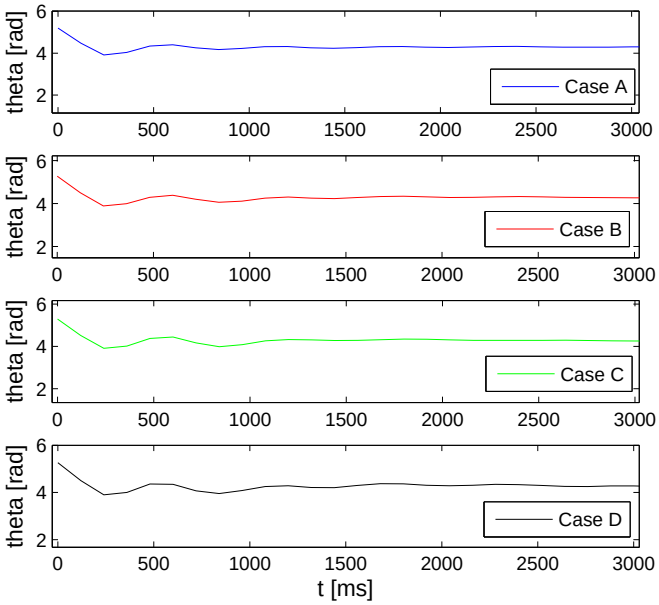

Figure 52: First instants of the evolution in time of the θ coordinate, for different values of N_p and N_c.

9 EBCUF experimentation

As it was mentioned in the introduction, the experimentation with the Estimator Based Control Using Flatness is the contribution of this project. In this section, the experimental results obtained are presented. Not only the most satisfactory ones but also the issues that have appeared during the implementation are explained.

The first observation made during the experiments was that the system was quite sensitive to the choice of poles for the matrices K_x, K_y, L_s and L_y. This is something that did not appear in simulation. The reason to obtain this behavior could be the constraints of the experimental setup which are explained as follows.

Constraints of the experimental setup:

It has been previously explained that the measurements were obtained by using a camera and using LabVIEW as interface and to communicate with the robot. The experiments shown that a feasible sampling period for the implementation of the EBCUF was $T_s = 160ms$. The approximated percentages of this sampling period needed for the different tasks are:

- Image acquisition and conversion of it into a matrix of pixels (33%)
- Image treatment to get the state measurements, filtering of these measures with a Kalman filter and solution of the EBCUF algorithm to get the inputs: speeds of both wheels (33%)
- Communication with the robot to apply the inputs (33%)

Actually these percentages of T_s give the longest times these tasks need to be done. However, the sampling period needs to be fixed taking into account the worst conditions.

Having a long sampling period has the advantage of having a filtering effect, however it is not desirable due to the following reasons.

By Shannon's theorem, we know that to be able to reconstruct a continuous signal using its samples, we need to get these samples with a frequency greater than the double of the highest frequency appearing in the initial signal. In practice, this theoretical value is recommended to be larger, i.e., it is desirable that the sampling frequency is from 10 to 20 times greater than the highest frequency in the initial signal.

This theorem implies that to be able to know the state evolution from the measured or filtered values, we need the highest frequency in the state dynamics to be less than a half of our sampling frequency. Consequently **low** speeds are required.

Another issue to be considered is the integration which is done in the EBCUF. The dynamics of the EBCUF is faster or slower depending on the magnitude of the control and observation poles. If these poles are chosen too large comparing to the long sampling period we have, the EBCUF dynamics is too fast to make the integration by Euler accurate enough. Therefore, the system becomes unstable.

When using the low speeds required by the sampling period, the problem that appears is that the motors of the Khepera work in the region of **dry friction**. It means the inputs have an effect which is smaller than the expected one.

Another constraint in the experiments is that the robot only accepts integer values for the inputs, i.e., the number of $pulses/ms$ sent to each motor. This introduces an error because not applying the real consign makes a difference of speeds in the wheels which was not explicitly given by the controller. This difference of speeds introduces a not desired increment on the angular speed because

$$\omega = \frac{V_r - V_l}{L} \tag{76}$$

The approach to this problem consists on having two loops with different sampling periods:

- A slower loop ($T_s = 160ms$) for the measurements, filtering and control algorithm
- A faster loop for the communication with the robot

This faster loop is given the inputs real values. Then, the idea of how it works is clearly reflected in this example: if the consign is 4.8 pulses to be sent during the longer sampling period ($T_s = 160ms$), during an 80% percent of it ($128ms$) a consign of 5 pulses is applied and the other 20% ($32ms$), the consign is 4 pulses. The problem is that changing the number of $pulses/ms$ sent to the wheels takes some time. Therefore, it could happen that the faster loop decides to apply a change in the number of pulses for one wheel at a precise instant and not be able to apply it because it is still sending the other wheel the signal to change it. It could also happen that the communication with the robot is still done when the longer sampling period finishes, which means new input values are needed. Hence if any change in the inputs needs to be applied, but it is not possible because the communication with the robot is taking place, this required change is done immediately after this communication finishes.

It has been observed that the results are more satisfactory including this faster loop, even if the idea of this approach cannot be exactly implemented as it has been thought.

Note:
For the experiments, 5 mm 'ball' was added around the final reference point, to stop the motors of both wheels of the robot once it entered the ball. It was needed to do that, because the noise in the filtered values gave always an error, sometimes real and sometimes unreal, that made the controller permanently give non-zero inputs.

9.1 Sensitivity of the system to the poles location for the EBCUF

In section 7, simulations with this control algorithm where analyzed. As it is presented in 7.4.1, double real poles from -0.5 to -30, constant for the calculation of the four matrices,K_x, K_y, L_s and L_y, offered good results.

However, in the experiments, too high poles made the system unstable and too low poles gave consign speeds too low to make the robot move.

Different experiments, all with a reference point equal to $[450mm, 450mm]$ were done. The first experiments tried to find the interval for the poles to present good results. The experiments with the following cases were done:

Case	$K_x - poles$	$K_y - poles$	$L_x - poles$	$L_y - poles$	K_p
Case A	$[-1, -1]$	$[-1, -1]$	$[-1, -1]$	$[-1, -1]$	15
Case B	$[-2, -2]$	$[-2, -2]$	$[-2, -2]$	$[-2, -2]$	15
Case C	$[-3, -3]$	$[-3, -3]$	$[-3, -3]$	$[-3, -3]$	15
Case D	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	$[-4, -4]$	15
Case E	$[-5, -5]$	$[-5, -5]$	$[-5, -5]$	$[-5, -5]$	15

Table 18: Poles chosen to study the effect of its magnitude in the experiments.

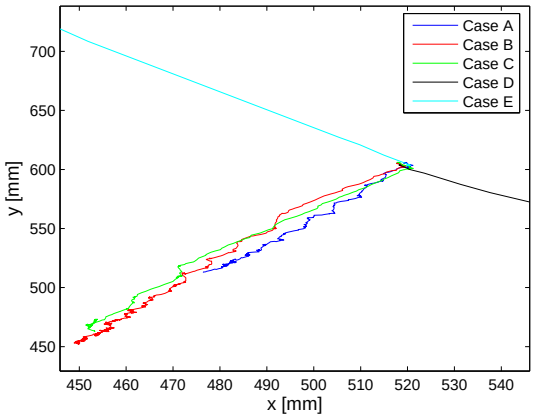

Figure 53: Trajectories followed with different magnitudes of the $x, y - poles$. Final reference point: $[450mm, 450mm]$.

In figure 53, we can see that case A does not reach the reference point, $[450mm, 450mm]$. This experiment corresponds to double real poles equal to -1, which seem not to be high enough to make the robot move. This simulation was stopped after the consign speeds of the robot were from 0 to 1 during almost a minute and the robot was doing small turns without getting closer to the final point.

Also in figure 53, we can observe that cases B and C, with double real poles at -2 and -3 respectively, do reach the reference point. Then, cases D and E, with poles at -4 and -5 seem to be too high and make the system unstable.

Hence, an approximate interval around -2 and -3, seems to be the one that the system accepts for the poles needed to calculate K_x, K_y, L_s and L_y, when they are chosen the same for the four matrices.

9.2 Effect of the magnitude of the $x, y - poles$

After trying to get the magnitudes of poles that provide a satisfactory behavior, it was tried to show the effect explained in 7.4.3, in the simulation section. This effect appears when the poles for the control of the x and y chains of integrators are chosen with different magnitude. Then, the effect is manifested by the fact that the robot follows different trajectories.

As it was also explained in 7.4.3, this difference in the magnitude of the poles allows to somehow control the trajectory followed and the final orientation, which are not explicitly determined. In simulation we can appreciate that choosing faster poles in x makes the system reduce faster the error in this component and consequently get to the final point with an orientation close to $\pi/2$ or to $3\pi/2$. On the other hand, faster poles in y, make the error in y reduced faster and therefore get to the final point with an orientation close to 0 or to π.

Experiments varying the magnitude of the poles for the x and y poles were tried. Many combinations were tried, and once again, problems appeared due to the either too low or too fast speeds. It was observed than in the case of low speeds the problem was quite often due to the oscillations the estimator presented in θ_{hat}. Then, to improve this behavior, the poles for the estimators in both chains were tried to be incremented. With this modification, it was a bit more likely that the robot would move and it was observed that θ_{hat} had a less oscillatory behavior and tracked faster the measured angle curve. The reason for that is that increasing the estimation poles makes the estimators converge faster to the system state.

With the estimation poles at -5, experiments with the following cases in table lvxypoles were implemented.

Case	$K_x - poles$	$K_y - poles$	$L_x - poles$	$L_y - poles$	K_p
Case A	$[-2, -2]$	$[-3, -3]$	$[-5, -5]$	$[-5, -5]$	15

Table 19: Poles chosen to study the effect of a difference in their magnitude for the x and y control.

Figure 54 presents the trajectory followed when the x control poles are fixed at -2 and the y control poles at -3. We can already see in this figure how the system approaches the final point with an angle closer to 0 rather to $\pi/2$, even if the initial error in y was bigger than the one in x (approximated initial position: $[320, 312]$). This confirms that with this difference in the poles, the system does not simply follow a straight line between the initial and final point.

To make it more evident, the orientation evolution in time has been plotted in figure 55. We can see than even if there are oscillations in the angle, they are around a mean of approximately $0.5rad$.

Still trying to show more clearly the effect of this difference between the x and y control poles, the proportional gain K_p was augmented from 15 to 20. The cases tried with this proportional gain and the estimation poles at -5 are showed in table

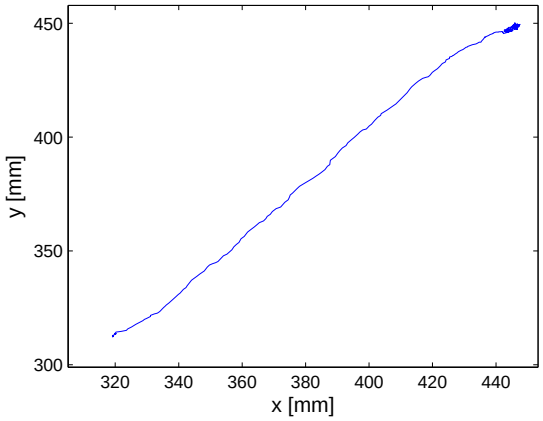

Figure 54: Trajectory followed in the cases that x control poles are-2 and y control poles are -3. Final reference point: $[450mm, 450mm]$.

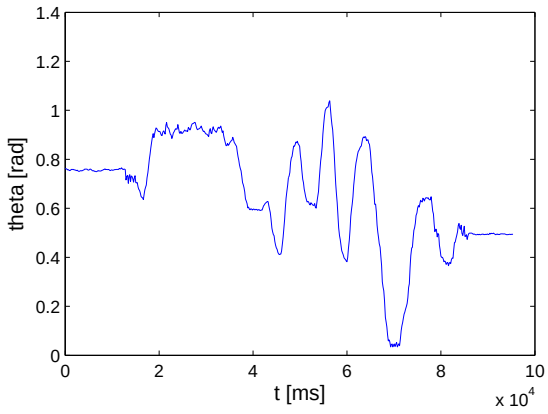

Figure 55: Evolution in time of the orientation, in the case that x control poles are-2 and y control poles are -3.

20.

Case	$K_x - poles$	$K_y - poles$	$L_x - poles$	$L_y - poles$	K_p
Case A	$[-2, -2]$	$[-3, -3]$	$[-5, -5]$	$[-5, -5]$	20
Case B	$[-3, -3]$	$[-3, -3]$	$[-5, -5]$	$[-5, -5]$	20
Case C	$[-3, -3]$	$[-2, -2]$	$[-5, -5]$	$[-5, -5]$	20

Table 20: Poles chosen to study the effect of a difference in their magnitude for the x and$y$ control with a proportional gain $K_p = 20$.

We can observe in figure 56 how the trajectory varies for the different control poles. This effect is evident even if the initial conditions are not exactly the same. We can confirm then that greater control poles in x reduce the error in this component faster than the one in y, and vice versa.

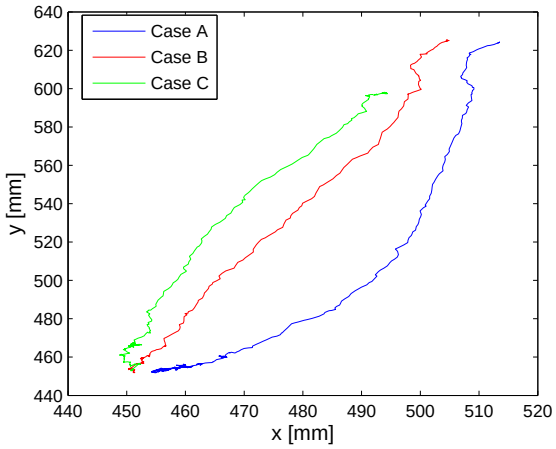

Figure 56: Trajectories followed with different x and y control poles and a proportional gain of $K_p = 20$. Final reference point: $[450mm, 450mm]$.

Then the final angle, can also be observed that is different for the three cases. As we said, faster poles in x will imply final angles closer to $\pi/2$ or $3\pi/2$ and faster poles in y will imply final angles closer too 0 and π. Figure 57 shows a part of figure 56 close to the final point $[450mm, 450mm]$. We can already see this effect of the poles on the final orientation. Moreover, the evolution in time of the orientation in each case has been plotted. We can see it in figure 58. The differences, specially between case B and C are not extremely clear. However, a tendency can be observed. The orientation in case A oscillates around 3.8, which is closer to π than to $3\pi/2$. In case B, it oscillates around 4.4; that is not far from the angle given by the straight line between the initial and final points, which is $4rad$ approximately. Finally the final orientation in case C oscillates around $4.7rad$ which is $3\pi/2rad$.

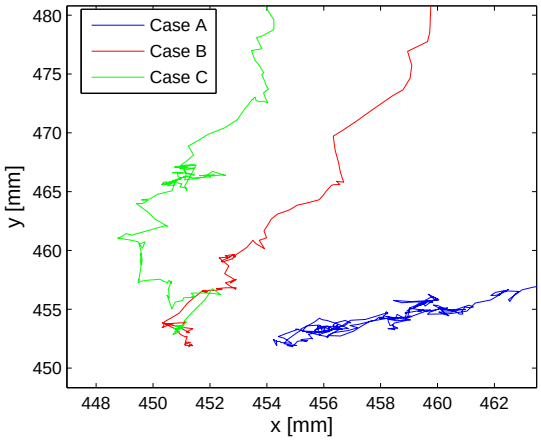

Figure 57: Region around the final point showing the trajectories followed with different x and y control poles and a proportional gain of $K_p = 20$. Final reference point: $[450mm, 450mm]$.

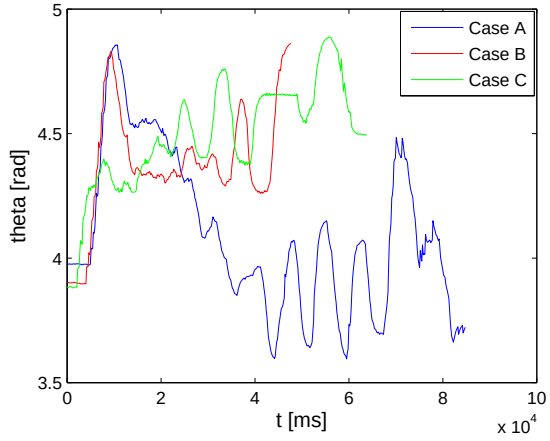

Figure 58: Evolution in time of the orientation, with different x and y control poles and a proportional gain of $K_p = 20$.

9.3 The noise problem

Something it can be observed in all the figures is that when the robot is close to the final point, quite big oscillations appear. This behavior is caused by the effect the noise has. The closest the robot is to the final point, the more important this effect becomes. This is caused by the fact that, close to the final point, the noise causes small errors in the position and/or orientation which imply a very different angle to be followed to reach the desired point. Therefore, this permanent noise produces big oscillations in θ_{hat} and consequently in the angular speed because θ_{hat} is used for the calculation of the consign of the angular speed.

Figure 59 presents $\theta_{filtered}$ and θ_{hat}, for a case with all the poles at -3 and $K_p = 20$. We observe that there are big oscillations at the first instants of the experiment because the estimator needs some time to converge into the filtered measurements of the state. After the estimator has converged, there is a region where both curves are quite similar. During this interval, we can observe that because the proportional controller is able to make θ follow θ_{hat}. Therefore, as it was explained in 4.3, the real system can be considered linear and this makes the EBCUF be stable. Then, a final region with big oscillations appears. This is caused, as the previous paragraph explains, to the filtered measurements noise.

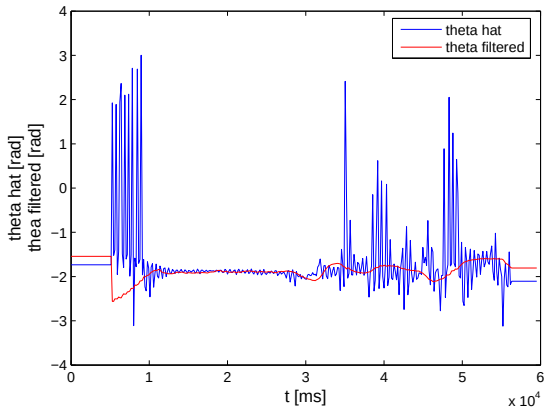

Figure 59: Evolution in time of the filtered measurement of the orientation ($\theta_{filtered}$) and the estimated orientation (θ_{hat}).

10 Conclusions and recommendations for future work

10.1 Conclusions

The experiments carried out during this project allow us to give the following conclusions.

Having a non-linear system, does not allow to get a robust control by applying a linear Model Predictive Control algorithm. When applying a linear control, approximations need to be done. The approximations presented in this project give a controller with a good performance for linear trajectories. This is achieved thanks to the on line linear reference trajectory which is used. The parameter setting of the MPC controller (weighting matrices in the cost function, size of the prediction and control horizons) does not require special tuning, since the results obtained are very similar for all the cases experimented.

The Estimator Based Control Using Flatness has provided good results. Something to highlight about it is the accuracy it was able to provide; deviations of 1 mm in the x and y coordinates respect to the reference point, which is better than what the MPC provided. However, this small deviations were got only after a relatively long interval of oscillations produced by the filtered measurements noise. It must also be said that this controller has shown a quite important sensitivity to the choice of poles due, in part, to the experimental setup constraints. Therefore the parameter tuning is much more important than in the MPC implemented.

Finally, the author's global opinion about this project is that it has included several theoretical aspects: position estimation and two types of control. So it has been very enriching in terms of academic formation.

10.2 Recommendations for future work

Concerning Model Predictive Control, the application of a non-linear controller minimizing the cost function chosen at every sampling period could be a very interesting work. It would be useful to compare it with the control implemented in this project and in [Herrera, 1999] and [Niederberger, 2002], which do approximations to apply a linear MPC.

Future work with an Estimator Based Control Using Flatness could be done by implementing a control with a reduced order estimator, since the components of the state x and y are already measured. Therefore the estimator would only estimate $\dot{x}$ and $\dot{y}$, and not x, y, $\dot{x}$ and $\dot{y}$ as it has been implemented in this project.

For both control strategies, other references could be experimented. Which would be really interesting for the case of the EBCUF, because the reference was not explicitly determined in the experiments carried out. Also a model including the inertia of the robot could be considered.

To improve the performance of the system when the speeds of the wheels are very low, a simple PI controller could be applied for each wheel, using the feedback given by the available odometric measurements that give the angle turned by the wheels. Doing this, the number of $pulses/ms$ sent to the motors to get a determined

speed would be variable and not constant as in this project and in [Herrera, 1999] and [Niederberger, 2002].

Experiments to get the dynamics of each motor, which are probably not the same, could also be useful.

For position estimation, other algorithms of image treatment and other filters could be applied in order to get better accuracy and faster sampling periods.

References

[Herrera, 1999] Herrera, Juan: *Trajectory tracking through predictive control.* Master thesis at Laboratoire d'Automatique(LA), École Polytechnique Fédérale de Lausanne(EPFL), Lausanne, July 1999.

[Niederberger, 2002] Niederberger, Andre: *Predictive control Design for a Khepera robot.* Master thesis at LA, EPFL, Lausanne, February 2002.

[Khepera manual, 1999] K-Team: *Khepera User Manual.* Lausanne, 1999.

[Camacho and Bordons, 1999] E.F.Camacho and C. Bordons: *Model Predictive Control.*Springer-Verlag London Limited 1999.

[Rawlings, 1999] James B. Rawlings:*Tutorial: Model Predictive Control Technology.* Department of Chemical Engineering.University of Wisconsin-Madison.Madison, WI 53706-1691. June 1999.

[Nikolaou, 2001] Michael Nikolaou:*Model Predictive Controllers: A Critical Synthesis of Theory and Industrial Needs.* Advances in Chemical Engineering Series, Academic Press (2001).

[Allgöwer et al., 2004] F. Allgöwer, R. Findeisen and Z.K. Nagy:*Nonlinear Model Predictive Control: From Theory to Application.* Institute for Systems Theory in Engineering, University of Stuttgart. 70550 Stuttgart, Germany 2004.

[Chi-Tsong Chen, 1999] Chi-Tsong Chen:*Linear System Theory and Design.* New York Oxford. Oxford University Press.

[D. Buccieri et al., 2005] D. Buccieri, Ph. Mullhaupt, Z.P. Jiang and D.Bonvin:*An output feedback controller for the global asymptotical stabilization of a class of nonholonomic systems.* Laboratoire d'Automatique(LA), École Polytechnique Fédérale de Lausanne(EPFL), Lausanne, 2005.

[Welch and Bishop, 2001] G. Welch and G. Bishop:*An Introduction to the Kalman Filter* University of North Carolina at Chapel Hill, Computer Science, February 2001.

[J.M. Miamo ,2005] J.M. Miamo:*Positioning and control of mobile robots.* Projet de semestre at LA, EPFL, Lausanne, June 2005.

[Khepera] K-Team: *Khepera.*
http://www.k-team.com/robots/khepera/index.html

[Seba, 2000] Seba, Oubouzid: *Commande prédictive de véhicules autonomes.* Travail Pratique de DEA. IA,EPFL, Lausanne, September 2000.

A Minimization of cost functions

In this section, a few mathematical results are derived that are needed for predictive control. In particular, cost function minimization is a central task in predictive control, and several cost function types are shown [Niederberger, 2002].

A.1 Functions of the type $J = (a - Bx)^T(a - Bx)$

Suppose a vector x has to be found which minimizes a given cost function J

$$J = (a - Bx)^T(a - Bx) \tag{77}$$

One way to find this vector is to set the first derivative of J with respect to x equal to zero

$$\frac{\partial J}{\partial x} = 2(a - Bx)^T(-B) = 0$$

transposing the equation yields

$$\begin{aligned} B^T(a - Bx) &= 0 \\ B^Ta - B^TBx &= 0 \\ B^TBx &= B^Ta \end{aligned}$$

and if the inverse of B^TB exists:

$$x = (B^TB)^{-1}B^Ta \tag{78}$$

The vector x minimizes the given cost function J.

A.2 Functions of the type $J = (a - Bx)^T(a - Bx) + x^T\Lambda x$

Suppose a vector x has to be found which minimizes a different cost function J, where Λ is called weighting matrix.

$$J = (a - Bx)^T(a - Bx) + x^T\Lambda x \tag{79}$$

Without loss of generality, Λ can be assumed to be symmetric. Every matrix can be split into a sum of a symmetric and a skew-symmetric part. But $x^T\Lambda_{skew-symmetric}x = 0$ so only the symmetric part has to be considered, that can be called Λ. In order to find the vector x that minimizes J, the derivation of J with respect to x is set equal to zero.

$$\frac{\partial J}{\partial x} = 2(a - Bx)^T(-B) + 2x^T\Lambda = 0$$

Transposing ($\Lambda^T = \Lambda$)

$$\begin{aligned} -B^T a + B^T Bx + \Lambda^T x &= 0 \\ (B^T B + \Lambda)x &= B^T a \end{aligned}$$

The weighting matrix Λ is chosen so that the inverse of $B^T B + \Lambda$ exists (for instance $\Lambda = \lambda I$ with $\lambda > 0$)

$$x = (B^T B + \Lambda)^{-1} B^T a \tag{80}$$

A.3 Functions of the type $J = (a - Bx)^T Q(a - Bx) + x^T \Lambda x$

Consider a cost function which allows to weigh $(a - Bx)$ with the matrix Q.

$$J = (a - Bx)^T Q(a - Bx) + x^T \Lambda x \tag{81}$$

With the same reasoning as in the previous section, the weighting matrix $Q = Q_r^T Q_r$ is a symmetric and positive-definite matrix. The same is true for $\Lambda = \Lambda_r^T \Lambda_r$.

$$\begin{aligned} J &= (a - Bx)^T Q(a - Bx) + x^T \Lambda x \\ &= (a - Bx)^T Q_r^T Q_r (a - Bx) + x^T \Lambda x \\ &= (Q_r(a - Bx))^T Q_r (a - Bx) + x^T \Lambda x \\ &= (Q_r a - Q_r Bx)^T (Q_r a - Q_r Bx) + x^T \Lambda x \end{aligned}$$

The vector x that minimizes the cost function is found by setting the derivative of J with respect to x equal to zero.

$$\frac{\partial J}{\partial x} = 2(Q_r a - Q_r Bx)^T (-Q_r B) + 2x^T \Lambda = 0$$

Transpose

$$\begin{aligned} (-Q_r B)^T (Q_r a - Q_r Bx) + \Lambda x &= 0 \\ -B^T Q_r^T Q_r a + B^T Q_r^T Q_r Bx + \Lambda x &= 0 \\ -B^T Q a + (B^T Q B + \Lambda)x &= 0 \end{aligned}$$

And again, choose Λ such that the inverse of $B^T QB + \Lambda$ exists:

$$x = (B^T QB + \Lambda)^{-1} B^T Qa \tag{82}$$

The vector x minimizes the given cost function J.

A.4 Functions of the type $J = (a-Bx)^T Q(a-Bx)+x^T \Lambda x+(c-Dx)^T \Gamma (c-Dx)$

Consider a cost function which in addition introduces the term $c - Dx$ which is weighted by Γ. This could be for instance the difference of a reference velocity to an predicted velocity.

$$J = (a - Bx)^T Q(a - Bx) + x^T \Lambda x + (c - Dx)^T \Gamma (c - Dx) \tag{83}$$

With the same reasoning as in the previous section, the weighting matrix $Q = Q_r^T Q_r$ is a symmetric and positive-definite matrix. The same is true for $\Lambda = \Lambda_r^T \Lambda_r$ and $\Gamma = \Gamma_r^T \Gamma_r$.

$$\begin{aligned}
J &= (a - Bx)^T Q(a - Bx) + x^T \Lambda x + (c - Dx)^T \Gamma (c - Dx) \\
&= (a - Bx)^T Q_r^T Q_r (a - Bx) + x^T \Lambda x + (c - Dx)^T \Gamma_r^T \Gamma_r (c - Dx) \\
&= (Q_r(a - Bx))^T Q_r (a - Bx) + x^T \Lambda x + (\Gamma_r(c - Dx))^T \Gamma_r (c - Dx) \\
&= (Q_r a - Q_r Bx)^T (Q_r a - Q_r Bx) + x^T \Lambda x + (\Gamma_r c - \Gamma_r Dx)^T (\Gamma_r c - \Gamma_r Dx)
\end{aligned}$$

The vector x that minimizes the cost function is found by setting the derivative of J with respect to x equal to zero.

$$\frac{\partial J}{\partial x} = 2(Q_r a - Q_r Bx)^T (-Q_r B) + 2x^T \Lambda + 2(\Gamma_r c - \Gamma_r Dx)^T (-\Gamma_r D) = 0$$

Transpose

$$\begin{aligned}
(-Q_r B)^T (Q_r a - Q_r Bx) + \Lambda x + (-\Gamma_r D)^T (\Gamma_r c - \Gamma_r Dx) &= 0 \\
-B^T Q_r^T Q_r a + B^T Q_r^T Q_r Bx + \Lambda x - D^T \Gamma_r^T \Gamma_r c + D^T \Gamma_r^T \Gamma_r Dx &= 0 \\
-B^T Qa - D^T \Gamma c + (B^T QB + \Lambda + D^T \Gamma D)x &= 0
\end{aligned}$$

And again, choose Λ such that the inverse of $B^T QB + \Lambda$ exists:

$$x = (B^T QB + \Lambda + D^T \Gamma D)^{-1} (B^T Qa + D^T \Gamma c) \tag{84}$$

The vector x minimizes the given cost function J.

B Derivation of various predictive control models

B.1 Derivation of the model in the form $\hat{\mathbf{x}} = \mathbf{Z}x(k) + \mathbf{F}\mathbf{u}(k)$

Suppose a dynamic model is given in the following form:

$$x(k+1) = \mathbf{A}x(k) + \mathbf{B}u(k) \tag{85}$$

where

$$\begin{aligned} x &\in \Re^n \\ u &\in \Re^m \\ \mathbf{A} &\in \Re^{n\times n} \\ \mathbf{B} &\in \Re^{n\times m} \end{aligned}$$

The state vector at time $k+2$ is calculated to be

$$\begin{aligned} x(k+2) &= \mathbf{A}x(k+1) + \mathbf{B}u(k+1) \\ &= \mathbf{A}^2x(k) + \mathbf{AB}u(k) + \mathbf{B}u(k+1) \end{aligned}$$

Since the values are estimated based on the knowledge available at time k

$$\hat{x}(k+2 \mid k) = \mathbf{A}^2x(k) + \mathbf{AB}u(k) + \mathbf{B}u(k+1)$$

And at time $k+3$, the state vector is

$$\begin{aligned} x(k+3) &= \mathbf{A}x(k+2) + \mathbf{B}u(k+2) \\ &= \mathbf{A}^3x(k) + \mathbf{A}^2\mathbf{B}u(k) + \mathbf{AB}u(k+1) + \mathbf{B}u(k+2) \end{aligned}$$

with the same reasoning as before

$$\hat{x}(k+3 \mid k) = \mathbf{A}^3x(k) + \mathbf{A}^2\mathbf{B}u(k) + \mathbf{AB}u(k+1) + \mathbf{B}u(k+2)$$

And in general at time $k+j$ with $j = 1, 2, \cdots, N_p$

$$\hat{x}(k+j \mid k) = \mathbf{A}^jx(k) + \mathbf{A}^{j-1}\mathbf{B}u(k) + \mathbf{A}^{j-2}\mathbf{B}u(k+1) + \cdots + \mathbf{B}u(k+j-1)$$

This can be written in matrix form as follows

$$\underbrace{\begin{pmatrix} \hat{x}(k+1 \mid k) \\ \hat{x}(k+2 \mid k) \\ \cdots \\ \hat{x}(k+N_p \mid k) \end{pmatrix}}_{\hat{\mathbf{x}}} = \underbrace{\begin{pmatrix} \mathbf{A}^1 \\ \mathbf{A}^2 \\ \cdots \\ \mathbf{A}^{N_p} \end{pmatrix}}_{\mathbf{Z}} x(k) + \underbrace{\begin{pmatrix} \mathbf{B} & 0 & \cdots & 0 \\ \mathbf{AB} & \mathbf{B} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{A}^{N_p-1}\mathbf{B} & \mathbf{A}^{N_p-2}\mathbf{B} & \cdots & \mathbf{B} \end{pmatrix}}_{\underline{\mathbf{F}}} \underbrace{\begin{pmatrix} u(k) \\ u(k+1) \\ \cdots \\ u(k+N_p-1) \end{pmatrix}}_{\mathbf{u}}$$

where

$$\begin{aligned}\hat{\mathbf{x}} &\in \Re^{nN_p}\\ \mathbf{Z} &\in \Re^{nN_p \times n}\\ \underline{\mathbf{F}} &\in \Re^{nN_p \times mN_p}\\ \mathbf{u} &\in \Re^{mN_p}\end{aligned}$$

The prediction of N_p samples of $\hat{\mathbf{x}}$ is done with the knowledge available at time k, assuming N_c samples of the control signal $\mathbf{u}$ are applied, and then kept constant. The control horizon is said to be N_c and the prediction horizon is N_p.

With the introduced notation

$$\hat{\mathbf{x}} = \mathbf{Z}x(k) + \underline{\mathbf{F}}\mathbf{u} \tag{86}$$

The matrix $\underline{\mathbf{F}}$ will be split into two submatrices $\overline{\mathbf{F}}$ and $\tilde{\mathbf{F}}$ such that

$$\underline{\mathbf{F}} = [\overline{\mathbf{F}} \mid \tilde{\mathbf{F}}]$$

where

$$\begin{aligned}\overline{\mathbf{F}} &\in \Re^{nN_p \times mN_c}\\ \tilde{\mathbf{F}} &\in \Re^{nN_p \times m(N_p - N_c)}\end{aligned}$$

The vector $\mathbf{u}$ is split into $\overline{\mathbf{u}}$ and $\tilde{\mathbf{u}}$ such that

$$\mathbf{u} = \left(\begin{array}{c} u(k) \\ u(k+1) \\ \cdots \\ \hline u(k+N_c-1) \\ u(k+N_c) \\ \cdots \\ u(k+N_c) \end{array}\right) = \left(\begin{array}{c} \overline{\mathbf{u}} \\ \hline \tilde{\mathbf{u}} \end{array}\right)$$

where

$$\begin{aligned}\overline{\mathbf{u}} &\in \Re^{mN_c}\\ \tilde{\mathbf{u}} &\in \Re^{m(N_p - N_c)}\end{aligned}$$

Split $\tilde{\mathbf{u}}$:

$$\begin{pmatrix} u(k+N_c) \\ u(k+N_c) \\ \cdots \\ u(k+N_c) \end{pmatrix} = \begin{pmatrix} \mathbf{I}_{m\times m}u(k+N_c) \\ \mathbf{I}_{m\times m}u(k+N_c) \\ \cdots \\ \mathbf{I}_{m\times m}u(k+N_c)) \end{pmatrix} = \begin{pmatrix} \mathbf{I}_{m\times m} \\ \mathbf{I}_{m\times m} \\ \cdots \\ \mathbf{I}_{m\times m} \end{pmatrix} u(k+N_c)$$

where

$$u(k+N_c) \in \Re^{m\times 1}$$

This allows to write

$$\hat{\mathbf{x}} = \mathbf{Z}x(k) + \overline{\mathbf{F}} \begin{pmatrix} u(k) \\ u(k+1) \\ \cdots \\ u(k+N_c-1) \end{pmatrix} + \tilde{\mathbf{F}} \underbrace{\begin{pmatrix} \mathbf{I}_{m\times m} \\ \mathbf{I}_{m\times m} \\ \cdots \\ \mathbf{I}_{m\times m} \end{pmatrix}}_{m(N_p-Nc)\times m} u(k+N_c)$$

Defining

$$\tilde{\tilde{\mathbf{F}}} = \tilde{\mathbf{F}} \begin{pmatrix} \mathbf{I}_{m\times m} \\ \mathbf{I}_{m\times m} \\ \cdots \\ \mathbf{I}_{m\times m} \end{pmatrix} \in \Re^{nN_p\times m}$$

writing

$$\hat{\mathbf{x}} = \mathbf{Z}x(k) + \overline{\mathbf{F}} \begin{pmatrix} u(k) \\ u(k+1) \\ \cdots \\ u(k+N_c-1) \end{pmatrix} + \tilde{\tilde{\mathbf{F}}} u(k+N_c)$$

$$\hat{\mathbf{x}} = \mathbf{Z}x(k) + [\overline{\mathbf{F}} \mid \tilde{\tilde{\mathbf{F}}}] \begin{pmatrix} u(k) \\ u(k+1) \\ \cdots \\ u(k+N_c) \end{pmatrix}$$

writing

$$\mathbf{F} = [\overline{\mathbf{F}} \mid \tilde{\tilde{\mathbf{F}}}]$$

allows to write the model in the form

$$\hat{\mathbf{x}} = \mathbf{Z}x(k) + \mathbf{F}\hat{\mathbf{u}}(k) \tag{87}$$

where

$$\hat{\mathbf{u}} = \begin{pmatrix} u(k) \\ u(k+1) \\ \cdots \\ u(k+N_c) \end{pmatrix} \in \Re^{m(N_c+1)}$$

$$\mathbf{F} \in \Re^{nN_p\times m(N_c+1)}$$

B.2 Derivation of the model in the form $\hat{\mathbf{x}}$ =$\mathbf{Z}x(k) + \mathbf{M}u(k-1) + \mathbf{F}\boldsymbol{\Delta}\mathbf{u}$

This model shows the *increment* Δu of the control signal. An integration effect and thus reduced steady state errors are expected.

A dynamic model

$$x(k+1) = \mathbf{A}x(k) + \mathbf{B}u(k) \tag{88}$$

where

$$\begin{aligned} x &\in \Re^n \\ u &\in \Re^m \\ \mathbf{A} &\in \Re^{n\times n} \\ \mathbf{B} &\in \Re^{n\times m} \end{aligned}$$

adding the terms $-\mathbf{B}u(k-1) + \mathbf{B}u(k-1)$

$$x(k+1) = \mathbf{A}x(k) + \mathbf{B}u(k) - \mathbf{B}u(k-1) + \mathbf{B}u(k-1)$$

Realizing that $\Delta u(k) = u(k) - u(k-1)$

$$x(k+1) = \mathbf{A}x(k) + \mathbf{B}\Delta u(k) + \mathbf{B}u(k-1)$$

Since the values are estimated based on the knowledge available at time k

$$\hat{x}(k+1 \mid k) = \mathbf{A}x(k) + \mathbf{B}\Delta u(k) + \mathbf{B}u(k-1)$$

The state vector at time $k+2$ is calculated to be

$$\begin{aligned} x(k+2) &= \mathbf{A}x(k+1) + \mathbf{B}u(k+1) - \mathbf{B}u(k) + \mathbf{B}u(k) - \mathbf{B}u(k-1) + \mathbf{B}u(k-1) \\ &= \mathbf{A}x(k+1) + \mathbf{B}\Delta u(k+1) + \mathbf{B}\Delta u(k) + \mathbf{B}u(k-1) \\ &= \mathbf{A}^2x(k) + \mathbf{AB}\Delta u(k) + \mathbf{AB}u(k-1) + \mathbf{B}\Delta u(k+1) + \mathbf{B}\Delta u(k) + \mathbf{B}u(k-1) \\ &= \mathbf{A}^2x(k) + (\mathbf{AB}+\mathbf{B})\Delta u(k) + \mathbf{B}\Delta u(k+1) + (\mathbf{AB}+\mathbf{B})u(k-1) \end{aligned}$$

with the same reasoning as before

$$\hat{x}(k+2 \mid k) = \mathbf{A}^2x(k) + (\mathbf{AB}+\mathbf{B})\Delta u(k) + \mathbf{B}\Delta u(k+1) + (\mathbf{AB}+\mathbf{B})u(k-1)$$

At time k+3

$$\begin{aligned}
x(k+3) &= \mathbf{A}x(k+2) + \mathbf{B}u(k+2) - \mathbf{B}u(k+1) + \mathbf{B}u(k+1) - \mathbf{B}u(k) + \mathbf{B}u(k) \\
&\quad -\mathbf{B}u(k-1) + \mathbf{B}u(k-1) \\
&= \mathbf{A}x(k+2) + \mathbf{B}\Delta u(k+2) + \mathbf{B}\Delta u(k+1) + \mathbf{B}\Delta u(k) + \mathbf{B}u(k-1) \\
&= \mathbf{A}^3x(k) + (\mathbf{A}^2\mathbf{B} + \mathbf{A}\mathbf{B})\Delta u(k) + \mathbf{A}\mathbf{B}\Delta u(k+1) + (\mathbf{A}^2\mathbf{B} + \mathbf{A}\mathbf{B})u(k-1) \\
&\quad +\mathbf{B}\Delta u(k+2) + \mathbf{B}\Delta u(k+1) + \mathbf{B}\Delta u(k) + \mathbf{B}u(k-1) \\
&= \mathbf{A}^3x(k) + (\mathbf{A}^2\mathbf{B} + \mathbf{A}\mathbf{B} + \mathbf{B})\Delta u(k) + (\mathbf{A}\mathbf{B} + \mathbf{B})\Delta u(k+1) + \mathbf{B}\Delta u(k+2) \\
&\quad +(\mathbf{A}^2\mathbf{B} + \mathbf{A}\mathbf{B} + \mathbf{B})u(k-1)
\end{aligned}$$

with the same reasoning as before

$$\begin{aligned}
\hat{x}(k+3 \mid k) = \mathbf{A}^3x(k) &+ (\mathbf{A}^2\mathbf{B} + \mathbf{A}\mathbf{B} + \mathbf{B})\Delta u(k) + (\mathbf{A}\mathbf{B} + \mathbf{B})\Delta u(k+1) + \mathbf{B}\Delta u(k+2) \\
&+(\mathbf{A}^2\mathbf{B} + \mathbf{A}\mathbf{B} + \mathbf{B})u(k-1)
\end{aligned}$$

And in general at time k+j

$$\begin{aligned}
\hat{x}(k+j \mid k) = \mathbf{A}^j x(k) &+ (\mathbf{A}^{j-1}\mathbf{B} + \cdots + \mathbf{A}^0\mathbf{B})\Delta u(k+0) + (\mathbf{A}^{j-2}\mathbf{B} + \cdots + \mathbf{A}^0\mathbf{B})\Delta u(k+1) \\
&+ \cdots + \mathbf{A}^0\mathbf{B}\Delta u(k+j-1) + (\mathbf{A}^{j-1}\mathbf{B} + \mathbf{A}^{j-2}\mathbf{B} + \cdots + \mathbf{A}^0\mathbf{B})u(k-1)
\end{aligned}$$

This can be written in matrix form

$$\underbrace{\begin{pmatrix} \hat{x}(k+1 \mid k) \\ \hat{x}(k+2 \mid k) \\ \cdots \\ \hat{x}(k+N_p \mid k) \end{pmatrix}}_{\hat{\mathbf{x}}} = \underbrace{\begin{pmatrix} \mathbf{A}^1 \\ \mathbf{A}^2 \\ \cdots \\ \mathbf{A}^{N_p} \end{pmatrix}}_{\mathbf{Z}} x(k) + \underbrace{\begin{pmatrix} \mathbf{B} \\ \mathbf{A}\mathbf{B} + \mathbf{B} \\ \mathbf{A}^2\mathbf{B} + \mathbf{A}\mathbf{B} + \mathbf{B} \\ \cdots \\ \mathbf{A}^{N_p-1}\mathbf{B} + \mathbf{A}^{N_p-2}\mathbf{B} + \cdots + \mathbf{A}^0\mathbf{B} \end{pmatrix}}_{\mathbf{M}} u(k-1) +$$

$$\underbrace{\begin{pmatrix} \mathbf{B} & 0 & \cdots & 0 \\ \mathbf{A}\mathbf{B} + \mathbf{B} & \mathbf{B} & \cdots & 0 \\ \mathbf{A}^2\mathbf{B} + \mathbf{A}\mathbf{B} + \mathbf{B} & \mathbf{A}\mathbf{B} + \mathbf{B} & \cdots & 0 \\ \cdots & \cdots & \cdots & 0 \\ \mathbf{A}^{N_p-1}\mathbf{B} + \cdots + \mathbf{A}^0\mathbf{B} & \mathbf{A}^{N_p-2}\mathbf{B} + \cdots + \mathbf{A}^0\mathbf{B} & \cdots & \mathbf{B} \end{pmatrix}}_{\underline{\mathbf{F}}} \cdot \underbrace{\begin{pmatrix} \Delta u(k) \\ \Delta u(k+1) \\ \Delta u(k+2) \\ \cdots \\ \Delta u(k+N_p-1) \end{pmatrix}}_{\boldsymbol{\Delta}\mathbf{u}}$$

where

$$\begin{aligned}
\hat{\mathbf{x}} &\in \Re^{nN_p} \\
\mathbf{Z} &\in \Re^{nN_p \times n} \\
\mathbf{M} &\in \Re^{nN_p \times m} \\
\underline{\mathbf{F}} &\in \Re^{nN_p \times mN_p} \\
\boldsymbol{\Delta}\mathbf{u} &\in \Re^{mN_p}
\end{aligned}$$

B.2 Derivation of the model in the form $\hat{\mathbf{x}} = \mathbf{Z}x(k) + \mathbf{M}u(k-1) + \mathbf{F}\Delta\mathbf{u}$

The prediction of N_p samples of $\hat{\mathbf{x}}$ is done with the knowledge available at time k, assuming N_c samples of the control signal $\mathbf{u}$ are applied, and then kept constant. Or, equivalently, $\Delta\mathbf{u} = 0$ after N_c. The control horizon is said to be N_c and the prediction horizon is N_p.
With the introduced notation

$$\hat{\mathbf{x}} = \mathbf{Z}x(k) + \mathbf{M}u(k-1) + \underline{\mathbf{F}}\Delta\mathbf{u} \tag{89}$$

The matrix $\underline{\mathbf{F}}$ will be split into two submatrices $\overline{\mathbf{F}}$ and $\tilde{\mathbf{F}}$ such that

$$\underline{\mathbf{F}} = [\overline{\mathbf{F}} \mid \tilde{\mathbf{F}}]$$

where

$$\overline{\mathbf{F}} \in \Re^{nN_p \times m(N_c+1)}$$
$$\tilde{\mathbf{F}} \in \Re^{nN_p \times m(N_p-N_c-1)}$$

Since it is assumed that after N_c samples of the control vector the increment of $\mathbf{u}$ is zero, split the vector $\Delta\mathbf{u}$ into $\Delta\hat{\mathbf{u}}$ and $\Delta\tilde{\mathbf{u}}$ such that

$$\Delta\mathbf{u} = \begin{pmatrix} \Delta\hat{\mathbf{u}} \\ \Delta\tilde{\mathbf{u}} \end{pmatrix} = \begin{pmatrix} \Delta u(k) \\ \Delta u(k+1) \\ \cdots \\ \Delta u(k+N_c) \\ \hline 0 \\ \cdots \\ 0 \end{pmatrix} = \begin{pmatrix} \Delta\hat{\mathbf{u}} \\ 0 \end{pmatrix}$$

where

$$\Delta\hat{\mathbf{u}} \in \Re^{mN_c+1}$$
$$\Delta\tilde{\mathbf{u}} \in \Re^{m(N_p-N_c)-1}$$

This allows to write ($\tilde{\mathbf{F}}0 = 0$)

$$\hat{x} = \mathbf{Z}x(k) + \mathbf{M}u(k-1) + \overline{\mathbf{F}} \begin{pmatrix} \Delta u(k) \\ \Delta u(k+1) \\ \cdots \\ \Delta u(k+N_c) \end{pmatrix}$$

realizing that

$$\underline{\mathbf{F}}\Delta\mathbf{u} = \overline{\mathbf{F}}\Delta\hat{\mathbf{u}}$$

defining

$$\mathbf{F} = [\overline{\mathbf{F}}]$$

results in

$$\hat{x} = \mathbf{Z}x(k) + \mathbf{M}u(k-1) + \mathbf{F}\Delta\hat{\mathbf{u}} \tag{90}$$

B.3 Derivation of the model in the form $\hat{\mathbf{x}} = \mathbf{Z}x(k) + \mathbf{M}u(k-1) + \mathbf{F}\boldsymbol{\Delta}\hat{\mathbf{u}} + \hat{\mathbf{e}}$

Take again Equation (88), but this time $\hat{x}(k)$ is predicted with the knowledge available at point k (or k-1):

$$\hat{x}(k \mid k) = \mathbf{A}x(k-1) + \mathbf{B}u(k-1)$$

However, the state vector $x(k)$ can be *measured* directly at point k. This will usually (if the model is not perfect) lead to a small error $e(k) = x(k) - \hat{x}(k \mid k)$. To get more precise values, it might be a good idea to *add* this error $e(k)$ to the future predictions of $\hat{x}(k+j \mid k) \quad j = 1 + \cdots + N_p$.

New predicted values of $\hat{x}$ are obtained

$$\hat{\mathbf{x}} = \mathbf{Z}x(k) + \mathbf{M}u(k-1) + \underline{\mathbf{F}}\boldsymbol{\Delta}\hat{\mathbf{u}} + \hat{\mathbf{e}} \tag{91}$$

where

$$\hat{\mathbf{e}} = \begin{pmatrix} I_{n\times n} \\ I_{n\times n} \\ \cdots \\ I_{n\times n} \end{pmatrix} e(k)$$

B.4 Derivation of the augmented system in the form $\hat{\mathbf{x}} = \mathbf{Z}(x)\mathbf{x}(k) + \mathbf{F}(k)\hat{\mathbf{u}}$

The equation of the system is given by

$$\mathbf{x}(k+1) = \mathbf{A}(k)\mathbf{x}(k) + \mathbf{B}(k)\mathbf{u}(k) \tag{92}$$

where

$$\begin{aligned} \mathbf{x} &\in \Re^n \\ \mathbf{u} &\in \Re^m \\ \mathbf{A}(k) &\in \Re^{n\times n} \\ \mathbf{B}(k) &\in \Re^{n\times m} \end{aligned}$$

At time $k+2$

$$\begin{aligned} \mathbf{x}(k+2) &= \mathbf{A}(k+1)\mathbf{x}(k+1) + \mathbf{B}(k+1)\mathbf{u}(k+1) \\ &= \mathbf{A}(k+1)\mathbf{A}(k)\mathbf{x}(k) + \mathbf{A}(k+1)\mathbf{B}(k)\mathbf{u}(k) + \mathbf{B}(k+1)\mathbf{u}(k+1) \end{aligned}$$

Since $\mathbf{x}(k+2)$ is estimated based on the knowledge available at time k

$$\hat{\mathbf{x}}(k+2 \mid k) = \mathbf{A}(k+1)\mathbf{A}(k)\mathbf{x}(k) + \mathbf{A}(k+1)\mathbf{B}(k)\mathbf{u}(k) + \mathbf{B}(k+1)\mathbf{u}(k+1)$$

And at time $k+3$

$$\begin{aligned}\mathbf{x}(k+3) &= \mathbf{A}(k+2)\mathbf{x}(k+2) + \mathbf{B}(k+2)\mathbf{u}(k+2)\\ &= \mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{A}(k)\mathbf{x}(k) + \mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{B}(k)\mathbf{u}(k)\\ &\quad +\mathbf{A}(k+2)\mathbf{B}(k+1)\mathbf{u}(k+1) + \mathbf{B}(k+2)\mathbf{u}(k+2)\end{aligned}$$

with the same reasoning as before

$$\begin{aligned}\hat{\mathbf{x}}(k+3 \mid k) &= \mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{A}(k)\mathbf{x}(k) + \mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{B}(k)\mathbf{u}(k)\\ &\quad +\mathbf{A}(k+2)\mathbf{B}(k+1)\mathbf{u}(k+1) + \mathbf{B}(k+2)\mathbf{u}(k+2)\end{aligned}$$

This can be written in matrix form as follows

$$\underbrace{\begin{pmatrix}\hat{\mathbf{x}}(k+1 \mid k)\\ \hat{\mathbf{x}}(k+2 \mid k)\\ \cdots\\ \hat{\mathbf{x}}(k+N_p \mid k)\end{pmatrix}}_{\hat{\mathbf{x}}} = \underbrace{\begin{pmatrix}\mathbf{A}(k)\\ \mathbf{A}(k+1)\mathbf{A}(k)\\ \mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{A}(k)\\ \cdots\\ \mathbf{A}(k+N_p-1)\cdots\mathbf{A}(k+1)\mathbf{A}(k)\end{pmatrix}}_{\mathbf{Z}(k)}\mathbf{x}(k)$$

$$+\underbrace{\begin{pmatrix}\mathbf{B}(k) & 0 & \cdots & \cdots & 0\\ \mathbf{A}(k+1)\mathbf{B}(k) & \mathbf{B}(k+1) & \cdots & \cdots & 0\\ \mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{B}(k) & \mathbf{A}(k+2)\mathbf{B}(k+1) & \cdots & \cdots & 0\\ \cdots & \cdots & \cdots & \mathbf{B}(k+N_p-2) & 0\\ \mathbf{A}(k+N_p-1)\cdots\mathbf{A}(k+1)\mathbf{B}(k) & \mathbf{A}(k+N_p-1)\cdots\mathbf{A}(k+2)\mathbf{B}(k+1) & \cdots & \mathbf{A}(k+N_p-1)\mathbf{B}(k+N_p-2) & \mathbf{B}(k+N_p-1)\end{pmatrix}}_{\underline{\mathbf{F}}(k)}\underbrace{\begin{pmatrix}\mathbf{u}(k)\\ \mathbf{u}(k+1)\\ \cdots\\ \mathbf{u}(k+N_p-1)\end{pmatrix}}_{\mathbf{u}}$$

where

$$\begin{aligned}\hat{\mathbf{x}} &\in \Re^{nN_p}\\ \mathbf{Z}(k) &\in \Re^{nN_p \times n}\\ \underline{\mathbf{F}}(k) &\in \Re^{nN_p \times mN_p}\\ \mathbf{u} &\in \Re^{mN_p}\end{aligned}$$

From this point, the operations are similar to the case where $\mathbf{Z}(k)$ and $\underline{\mathbf{F}}(k)$ were constant.
The matrix $\underline{\mathbf{F}}(k)$ will be split into two submatrices $\overline{\mathbf{F}}(k)$ and $\tilde{\mathbf{F}}(k)$ such that

$$\underline{\mathbf{F}}(k) = [\overline{\mathbf{F}}(k) \mid \tilde{\mathbf{F}}(k)]$$

where

$$\begin{aligned}\overline{\mathbf{F}}(k) &\in \Re^{nN_p \times mN_c}\\ \tilde{\mathbf{F}}(k) &\in \Re^{nN_p \times m(N_p - N_c)}\end{aligned}$$

The vector $\mathbf{u}$ is split into $\overline{\mathbf{u}}$ and $\tilde{\mathbf{u}}$ such that

$$\mathbf{u} = \begin{pmatrix} \mathbf{u}(k) \\ \mathbf{u}(k+1) \\ \cdots \\ \mathbf{u}(k+N_c-1) \\ \hline \mathbf{u}(k+N_c) \\ \cdots \\ \mathbf{u}(k+N_c) \end{pmatrix} = \begin{pmatrix} \overline{\mathbf{u}} \\ \hline \tilde{\mathbf{u}} \end{pmatrix}$$

where

$$\overline{\mathbf{u}} \in \Re^{mN_c}$$
$$\tilde{\mathbf{u}} \in \Re^{m(N_p-N_c)}$$

The vector $\tilde{\mathbf{u}}$ is split into

$$\begin{pmatrix} \mathbf{u}(k+N_c) \\ \mathbf{u}(k+N_c) \\ \cdots \\ \mathbf{u}(k+N_c) \end{pmatrix} = \begin{pmatrix} \mathbf{I}_{m\times m}\mathbf{u}(k+N_c) \\ \mathbf{I}_{m\times m}\mathbf{u}(k+N_c) \\ \cdots \\ \mathbf{I}_{m\times m}\mathbf{u}(k+N_c)) \end{pmatrix} = \begin{pmatrix} \mathbf{I}_{m\times m} \\ \mathbf{I}_{m\times m} \\ \cdots \\ \mathbf{I}_{m\times m} \end{pmatrix} \mathbf{u}(k+N_c)$$

where

$$\mathbf{u}(k+N_c) \in \Re^{m\times 1}$$

This allows to write

$$\hat{\mathbf{x}} = \mathbf{Z}(k)\mathbf{x}(k) + \overline{\mathbf{F}}(k) \begin{pmatrix} \mathbf{u}(k) \\ \mathbf{u}(k+1) \\ \cdots \\ \mathbf{u}(k+N_c-1) \end{pmatrix} + \tilde{\mathbf{F}}(k) \begin{pmatrix} \mathbf{I}_{m\times m} \\ \mathbf{I}_{m\times m} \\ \cdots \\ \mathbf{I}_{m\times m} \end{pmatrix} \mathbf{u}(k+N_c)$$

Defining

$$\tilde{\tilde{\mathbf{F}}}(k) = \tilde{\mathbf{F}}(k) \begin{pmatrix} \mathbf{I}_{m\times m} \\ \mathbf{I}_{m\times m} \\ \cdots \\ \mathbf{I}_{m\times m} \end{pmatrix} \in \Re^{nN_p\times m}$$

yields

$$\hat{\mathbf{x}} = \mathbf{Z}(k)\mathbf{x}(k) + \overline{\mathbf{F}}(k) \begin{pmatrix} \mathbf{u}(k) \\ \mathbf{u}(k+1) \\ \cdots \\ \mathbf{u}(k+N_c-1) \end{pmatrix} + \tilde{\tilde{\mathbf{F}}}(k)\mathbf{u}(k+N_c)$$

$$\hat{\mathbf{x}} = \mathbf{Z}(k)\mathbf{x}(k) + [\overline{\mathbf{F}}(k) \mid \tilde{\tilde{\mathbf{F}}}(k)] \begin{pmatrix} \mathbf{u}(k) \\ \mathbf{u}(k+1) \\ \cdots \\ \mathbf{u}(k+N_c) \end{pmatrix}$$

writing

$$\mathbf{F}(k) = [\overline{\mathbf{F}}(k) \mid \tilde{\tilde{\mathbf{F}}}(k)]$$

the following result is obtained

$$\hat{\mathbf{x}} = \mathbf{Z}(x)\mathbf{x}(k) + \mathbf{F}(k)\hat{\mathbf{u}} \tag{93}$$

where

$$\hat{\mathbf{u}} = \begin{pmatrix} \mathbf{u}(k) \\ \mathbf{u}(k+1) \\ \cdots \\ \mathbf{u}(k+N_c) \end{pmatrix} \in \Re^{m(N_c+1)}$$

$$\mathbf{F}(k) \in \Re^{nN_p \times m(N_c+1)}$$

B.5 Derivation of the augmented system in the form $\hat{\mathbf{x}} = \mathbf{Z}(k)\mathbf{x} + \mathbf{M}(k)\mathbf{u}(k-1) + \mathbf{F}(k)\Delta\mathbf{u}$

The equation of the system is given by

$$\mathbf{x}(k+1) = \mathbf{A}(k)\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k)$$

where

$$\begin{aligned} \mathbf{x} &\in \Re^n \\ \mathbf{u} &\in \Re^m \\ \mathbf{A}(k) &\in \Re^{n \times n} \\ \mathbf{B} &\in \Re^{n \times m} \end{aligned}$$

In this case, the matrix $\mathbf{B}$ is kept constant, and not time varying because it makes it possible to develop the model in function of the input increments. For the case of a variable matrix $\mathbf{B(k)}$, the approximation of supposing it constant during a sample period, i.e., from instant k to $k+1$, needs to be done. Then the algorithm would be the one here presented substituting $\mathbf{B}$ in the same way it was done in appendix B.4.

Adding the terms $-\mathbf{B}\mathbf{u}(k-1) + \mathbf{B}\mathbf{u}(k-1)$

$$\mathbf{x}(k+1) = \mathbf{A}(k)\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) - \mathbf{B}\mathbf{u}(k-1) + \mathbf{B}\mathbf{u}(k-1)$$

Realizing that $\Delta\mathbf{u}(k) = \mathbf{u}(k) - \mathbf{u}(k-1)$

$$\mathbf{x}(k+1) = \mathbf{A}(k)\mathbf{x}(k) + \mathbf{B}\Delta\mathbf{u}(k) + \mathbf{B}\mathbf{u}(k-1)$$

Since the estimation is based on the knowledge available at time k

$$\hat{\mathbf{x}}(k+1 \mid k) = \mathbf{A}(k)\mathbf{x}(k) + \mathbf{B}\Delta\mathbf{u}(k) + \mathbf{B}\mathbf{u}(k-1)$$

B.5 Derivation of the augmented system in the form $\hat{\mathbf{x}} = \mathbf{Z}(k)\mathbf{x} + \mathbf{M}(k)\mathbf{u}(k-1) + \mathbf{F}(k)\Delta\mathbf{u}$

At time k+2

$$\begin{aligned}
\mathbf{x}(k+2) &= \mathbf{A}(k+1)\mathbf{x}(k+1) + \mathbf{B}\mathbf{u}(k+1) - \mathbf{B}\mathbf{u}(k) + \mathbf{B}\mathbf{u}(k) - \mathbf{B}\mathbf{u}(k-1) + \mathbf{B}\mathbf{u}(k-1) \\
&= \mathbf{A}(k+1)\mathbf{x}(k+1) + \mathbf{B}\Delta\mathbf{u}(k+1) + \mathbf{B}\Delta\mathbf{u}(k) + \mathbf{B}\mathbf{u}(k-1) \\
&= \mathbf{A}(k+1)\mathbf{A}(k)\mathbf{x}(k) + \mathbf{A}(k+1)\mathbf{B}\Delta\mathbf{u}(k) + \mathbf{A}(k+1)\mathbf{B}\mathbf{u}(k-1) \\
&\quad + \mathbf{B}\Delta\mathbf{u}(k+1) + \mathbf{B}\Delta\mathbf{u}(k) + \mathbf{B}\mathbf{u}(k-1) \\
&= \mathbf{A}(k+1)\mathbf{A}(k)\mathbf{x}(k) + (\mathbf{A}(k+1)\mathbf{B} + \mathbf{B})\Delta\mathbf{u}(k) + \mathbf{B}\Delta\mathbf{u}(k+1) \\
&\quad + (\mathbf{A}(k+1)\mathbf{B} + \mathbf{B})\mathbf{u}(k-1)
\end{aligned}$$

with the same reasoning as before

$$\begin{aligned}
\hat{\mathbf{x}}(k+2 \mid k) = \mathbf{A}(k+1)\mathbf{A}(k)\mathbf{x}(k) &+ (\mathbf{A}(k+1)\mathbf{B} + \mathbf{B})\Delta\mathbf{u}(k) + \mathbf{B}\Delta\mathbf{u}(k+1) \\
&+ (\mathbf{A}(k+1)\mathbf{B} + \mathbf{B})\mathbf{u}(k-1)
\end{aligned}$$

At time k+3

$$\begin{aligned}
\mathbf{x}(k+3) &= \mathbf{A}(k+2)\mathbf{x}(k+2) + \mathbf{B}\mathbf{u}(k+2) - \mathbf{B}\mathbf{u}(k+1) + \mathbf{B}\mathbf{u}(k+1) - \mathbf{B}\mathbf{u}(k) + \mathbf{B}\mathbf{u}(k) \\
&\quad - \mathbf{B}\mathbf{u}(k-1) + \mathbf{B}\mathbf{u}(k-1) \\
&= \mathbf{A}(k+2)\mathbf{x}(k+2) + \mathbf{B}\Delta\mathbf{u}(k+2) + \mathbf{B}\Delta\mathbf{u}(k+1) + \mathbf{B}\Delta\mathbf{u}(k) + \mathbf{B}\mathbf{u}(k-1) \\
&= \mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{A}(k)\mathbf{x}(k) + (\mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{B} + \mathbf{A}(k+2)\mathbf{B})\Delta\mathbf{u}(k) \\
&\quad + \mathbf{A}(k+2)\mathbf{B}\Delta\mathbf{u}(k+1) + \mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{B} + \mathbf{A}(k+2)\mathbf{B}\mathbf{u}(k-1) \\
&\quad + \mathbf{B}\Delta\mathbf{u}(k+2) + \mathbf{B}\Delta\mathbf{u}(k+1) + \mathbf{B}\Delta\mathbf{u}(k) + \mathbf{B}\mathbf{u}(k-1) \\
&= \mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{A}(k)\mathbf{x}(k) + (\mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{B} + \mathbf{A}(k+2)\mathbf{B} + \mathbf{B})\Delta\mathbf{u}(k) \\
&\quad + (\mathbf{A}(k+2)\mathbf{B} + \mathbf{B})\Delta\mathbf{u}(k+1) + \mathbf{B}\Delta\mathbf{u}(k+2) \\
&\quad + (\mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{B} + \mathbf{A}(k+2)\mathbf{B} + \mathbf{B})\mathbf{u}(k-1)
\end{aligned}$$

with the same reasoning as before

$$\begin{aligned}
\hat{\mathbf{x}}(k+3 \mid k) = \mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{A}(k)\mathbf{x}(k) &+ (\mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{B} + \mathbf{A}(k+2)\mathbf{B} + \mathbf{B})\Delta\mathbf{u}(k) \\
&+ (\mathbf{A}(k+2)\mathbf{B} + \mathbf{B})\Delta\mathbf{u}(k+1) + \mathbf{B}\Delta\mathbf{u}(k+2) \\
&+ (\mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{B} + \mathbf{A}(k+2)\mathbf{B} + \mathbf{B})\mathbf{u}(k-1)
\end{aligned}$$

This can be written in matrix form

B.5 Derivation of the augmented system in the form $\hat{\mathbf{x}} = \mathbf{Z}(k)\mathbf{x} + \mathbf{M}(k)\mathbf{u}(k-1) + \mathbf{F}(k)\Delta\mathbf{u}$

$$\underbrace{\begin{pmatrix} \hat{\mathbf{x}}(k+1 \mid k) \\ \hat{\mathbf{x}}(k+2 \mid k) \\ \cdots \\ \hat{\mathbf{x}}(k+N_p \mid k) \end{pmatrix}}_{\hat{\mathbf{x}}} = \underbrace{\begin{pmatrix} \mathbf{A}(k) \\ \mathbf{A}(k+1)\mathbf{A}(k) \\ \cdots \\ \mathbf{A}(k+N_p-1)\cdots\mathbf{A}(k) \end{pmatrix}}_{\mathbf{Z}(k)} \mathbf{x}(k)$$

$$+ \underbrace{\begin{pmatrix} \mathbf{B} \\ \mathbf{A}(k+1)\mathbf{B}+\mathbf{B} \\ \mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{B}+\mathbf{A}(k+2)\mathbf{B}+\mathbf{B} \\ \cdots \\ \mathbf{A}(k+N_p-1)\cdots\mathbf{A}(k+1)\mathbf{B}+\cdots+\mathbf{A}(k+N_p-1)\mathbf{B}+\mathbf{B} \end{pmatrix}}_{\mathbf{M}(k)} \mathbf{u}(k-1)$$

$$+ \underbrace{\begin{pmatrix} \mathbf{B} & 0 & \cdots & 0 \\ \mathbf{A}(k+1)\mathbf{B}+\mathbf{B} & \mathbf{B} & \cdots & 0 \\ \mathbf{A}(k+2)\mathbf{A}(k+1)\mathbf{B}+\mathbf{A}(k+2)\mathbf{B}+\mathbf{B} & \mathbf{A}(k+2)\mathbf{B}+\mathbf{B} & \cdots & 0 \\ \cdots & & & \\ \mathbf{A}(k+N_p-1)\cdots\mathbf{A}(k+1)\mathbf{B}+\cdots+\mathbf{B} & \mathbf{A}(k+N_p-1)\cdots\mathbf{A}(k+2)\mathbf{B}+\cdots+\mathbf{B} & \cdots & \mathbf{B} \end{pmatrix}}_{\underline{\mathbf{F}}(k)} \underbrace{\begin{pmatrix} \Delta\mathbf{u}(k) \\ \Delta\mathbf{u}(k+1) \\ \Delta\mathbf{u}(k+2) \\ \cdots \\ \Delta\mathbf{u}(k+N_p-1) \end{pmatrix}}_{\Delta\mathbf{u}}$$

where

$$\begin{aligned} \hat{\mathbf{x}} &\in \Re^{nN_p} \\ \mathbf{Z}(k) &\in \Re^{nN_p \times n} \\ \mathbf{M}(k) &\in \Re^{nN_p \times m} \\ \underline{\mathbf{F}}(k) &\in \Re^{nN_p \times mN_p} \\ \Delta\mathbf{u} &\in \Re^{mN_p} \end{aligned}$$

From this point, the operations are similar to the case where $\mathbf{Z}(k)$, $\mathbf{M}(k)$ and $\underline{\mathbf{F}}(k)$ were constant.

With the introduced notation

$$\hat{\mathbf{x}} = \mathbf{Z}(k)\mathbf{x}(k) + \mathbf{M}(k)\mathbf{u}(k-1) + \underline{\mathbf{F}}(k)\Delta\mathbf{u} \tag{94}$$

B.5 Derivation of the augmented system in the form $\hat{\mathbf{x}} = \mathbf{Z}(k)\mathbf{x} + \mathbf{M}(k)\mathbf{u}(k-1) + \mathbf{F}(k)\Delta\mathbf{u}$

The matrix $\underline{\mathbf{F}}(k)$ will be split into two submatrices $\overline{\mathbf{F}}(k)$ and $\tilde{\mathbf{F}}(k)$ such that

$$\underline{\mathbf{F}}(k) = [\overline{\mathbf{F}}(k) \mid \tilde{\mathbf{F}}(k)]$$

where

$$\begin{aligned} \overline{\mathbf{F}}(k) &\in \Re^{nN_p \times mN_c - 1} \\ \tilde{\mathbf{F}}(k) &\in \Re^{nN_p \times m(N_p - N_c)+1} \end{aligned}$$

Since it is assumed that after N_c samples of the control vector the increment of $\mathbf{u}$ is zero, the vector $\Delta\mathbf{u}$ is split into $\Delta\hat{\mathbf{u}}$ and $\Delta\tilde{\mathbf{u}}$ such that

$$\Delta\mathbf{u} = \begin{pmatrix} \Delta\hat{\mathbf{u}} \\ \hline \Delta\tilde{\mathbf{u}} \end{pmatrix} = \left(\begin{array}{c} \Delta\mathbf{u}(k) \\ \Delta\mathbf{u}(k+1) \\ \cdots \\ \Delta\mathbf{u}(k+N_c) \\ \hline 0 \\ \cdots \\ 0 \end{array}\right) = \begin{pmatrix} \Delta\hat{\mathbf{u}} \\ \hline 0 \end{pmatrix}$$

where

$$\begin{aligned} \Delta\hat{\mathbf{u}} &\in \Re^{mN_c+1} \\ \Delta\tilde{\mathbf{u}} &\in \Re^{m(N_p-N_c)-1} \end{aligned}$$

This allows to write ($\tilde{\mathbf{F}}(k)0 = 0$)

$$\hat{\mathbf{x}} = \mathbf{Z}(k)\mathbf{x}(k) + \mathbf{M}(k)\mathbf{u}(k-1) + \overline{\mathbf{F}}(k) \begin{pmatrix} \Delta\mathbf{u}(k) \\ \Delta\mathbf{u}(k+1) \\ \cdots \\ \Delta\mathbf{u}(k+N_c) \end{pmatrix}$$

realizing that

$$\underline{\mathbf{F}}(k)\Delta\mathbf{u} = \overline{\mathbf{F}}(k)\Delta\hat{\mathbf{u}}$$

defining

$$\mathbf{F}(k) = [\overline{\mathbf{F}}(k)]$$

and so the result is obtained

$$\hat{\mathbf{x}} = \mathbf{Z}(k)\mathbf{x}(\mathbf{k}) + \mathbf{M}(k)\mathbf{u}(k-1) + \mathbf{F}(k)\Delta\hat{\mathbf{u}} \tag{95}$$

C Derivation of various control algorithms

Control algorithms are derived in this section for the models shown in the previous section.

C.1 Predictive control algorithm for a model in the form $\hat{\mathbf{x}} = \mathbf{Z}x(k) + \mathbf{F}u(k-1)$, called predictive model 1

A predictive control algorithm for this model is now derived, i.e. a sequence of control signals, of which only the first sample is applied to the system. The cost function which has to be minimized is defined as

$$\mathbf{J_1} = (\mathbf{r} - \hat{\mathbf{x}})^{\mathbf{T}}\mathbf{Q}(\mathbf{r} - \hat{\mathbf{x}}) + \hat{\mathbf{u}}^{\mathbf{T}}\mathbf{\Lambda}\hat{\mathbf{u}} \tag{96}$$

where

$$\mathbf{r} = \begin{pmatrix} r(k+1) \\ \cdots \\ r(k+N_p) \end{pmatrix} \in \Re^{nN_p}$$

$$\hat{\mathbf{u}} = \begin{pmatrix} \hat{u}(k) \\ \hat{u}(k+1) \\ \cdots \\ \hat{u}(k+N_c) \end{pmatrix} \in \Re^{m(N_c+1)}$$

$$\hat{\mathbf{x}} = \begin{pmatrix} \hat{x}(k+1) \\ \cdots \\ \hat{x}(k+N_p) \end{pmatrix} \in \Re^{nN_p}$$

$\mathbf{r}$ is the vector that contains the reference points, $\hat{\mathbf{u}}$ is the input vector, $\mathbf{Q}$ and $\mathbf{\Lambda}$ are symmetric and positive definite weighting matrices.
With the introduced notation

$$\mathbf{J_1} = \underbrace{(\mathbf{r} - \mathbf{Z}x(k)}_{a} - \underbrace{\mathbf{F}\hat{\mathbf{u}})}_{Bx}{}^T\mathbf{Q}(\mathbf{r} - \mathbf{Z}x(k) - \mathbf{F}\hat{\mathbf{u}}) + \underbrace{\hat{\mathbf{u}}^{\mathbf{T}}}_{x^T}\mathbf{\Lambda}\hat{\mathbf{u}}$$

A similarity with a function of the form $J = (a - \mathbf{B}x)^T Q(a - \mathbf{B}x) + x^T \Lambda x$ can be seen. This problem has been solved previously, so the vector $\hat{\mathbf{u}}$ is calculated according to the formula [82]:

$$\hat{\mathbf{u}} = (\mathbf{F}^{\mathbf{T}}\mathbf{Q}\mathbf{F} + \mathbf{\Lambda})^{-\mathbf{1}}\mathbf{F}^{\mathbf{T}}\mathbf{Q}(\mathbf{r} - \mathbf{Z}x(k))$$

Defining

$$\mathbf{K} = (\mathbf{F}^{\mathbf{T}}\mathbf{Q}\mathbf{F} + \mathbf{\Lambda})^{-\mathbf{1}}\mathbf{F}^{\mathbf{T}}\mathbf{Q} \in \Re^{m(N_c+1)\times nN_p} \tag{97}$$

$$\hat{\mathbf{x}}^{\mathbf{0}} = \mathbf{Z}x(k) \in \Re^{nN_p} \tag{98}$$

allows to write

$$\hat{\mathbf{u}} = \mathbf{K}(\mathbf{r} - \hat{\mathbf{x}}^{\mathbf{0}}) \tag{99}$$

Only the first line of $\hat{\mathbf{u}}$ is needed, so the first line of $\mathbf{K}$ is extracted, which is called $\mathbf{k^T}$.

$$\mathbf{K} = [\frac{\mathbf{k}^T}{\mathbf{K''}}] \text{ where } \mathbf{k}^T \in \Re^{m \times nN_p}$$

This means multiplying

$$\mathbf{k}^T = [\mathbf{I}_{m \times m} \mid \mathbf{0}_{m \times mN_c}]\mathbf{K}$$

The final result is

$$u(k) = \mathbf{k^T}(\mathbf{r} - \hat{\mathbf{x}}^{\mathbf{0}}) \tag{100}$$

C.2 Predictive control algorithm for a model in the form $\hat{\mathbf{x}} = \mathbf{Z}x(k) + \mathbf{M}u(k-1) + \mathbf{F}\boldsymbol{\Delta}\hat{\mathbf{u}}$

The predictive control law for this model is now derived. The cost function which has to be minimized is defined as

$$\mathbf{J} = (\mathbf{r} - \hat{\mathbf{x}})^{\mathbf{T}}\mathbf{Q}(\mathbf{r} - \hat{\mathbf{x}}) + \boldsymbol{\Delta}\hat{\mathbf{u}}^{\mathbf{T}}\boldsymbol{\Lambda}\boldsymbol{\Delta}\hat{\mathbf{u}}$$

where

$$\mathbf{r} = \begin{pmatrix} r(k+1) \\ \cdots \\ r(k+N_p) \end{pmatrix} \in \Re^{nN_p}$$

$$\boldsymbol{\Delta}\hat{\mathbf{u}} = \begin{pmatrix} \Delta\hat{u}(k) \\ \Delta\hat{u}(k+1) \\ \cdots \\ \Delta\hat{u}(k+N_c) \end{pmatrix} \in \Re^{m(N_c+1)}$$

$$\hat{\mathbf{x}} = \begin{pmatrix} \hat{x}(k+1) \\ \cdots \\ \hat{x}(k+N_p) \end{pmatrix} \in \Re^{nN_p}$$

The vector $\mathbf{r}$ contains the reference points, $\boldsymbol{\Delta}\hat{\mathbf{u}}$ is the input increment vector, $\mathbf{Q}$ and $\boldsymbol{\Lambda}$ are symmetric and positive definite weight matrices.
With the introduced notation

$$\mathbf{J} = \underbrace{(\mathbf{r} - \mathbf{Z}x(k) - \mathbf{M}u(k-1)}_{a} - \underbrace{\mathbf{F}\boldsymbol{\Delta}\hat{\mathbf{u}})}_{\mathbf{B}x}{}^{T}\mathbf{Q}(\mathbf{r} - \mathbf{Z}x(k) - \mathbf{M}u(k-1) - \mathbf{F}\boldsymbol{\Delta}\hat{\mathbf{u}}) + \underbrace{\boldsymbol{\Delta}\hat{\mathbf{u}}^{\mathbf{T}}}_{x^T}\boldsymbol{\Lambda}\boldsymbol{\Delta}\hat{\mathbf{u}}$$

A similarity with a function of the form $J = (a - \mathbf{B}x)^T\mathbf{Q}(a - \mathbf{B}x) + x^T\boldsymbol{\Lambda}x$ can be seen. This problem has been solved previously, so the vector $\boldsymbol{\Delta}\hat{\mathbf{u}}$ is calculated according to formula [82]:

$$\boldsymbol{\Delta}\hat{\mathbf{u}} = (\mathbf{F}^{\mathbf{T}}\mathbf{Q}\mathbf{F} + \boldsymbol{\Lambda})^{-\mathbf{1}}\mathbf{F}^{\mathbf{T}}\mathbf{Q}(\mathbf{r} - \mathbf{Z}x(k) - \mathbf{M}u(k-1))$$

Defining

$$\mathbf{K} = (\mathbf{F}^{\mathbf{T}}\mathbf{Q}\mathbf{F} + \boldsymbol{\Lambda})^{-\mathbf{1}}\mathbf{F}^{\mathbf{T}}\mathbf{Q} \in \Re^{m(N_c+1)\times nN_p} \quad (101)$$

$$\hat{\mathbf{x}}^{\mathbf{0}} = \mathbf{Z}x(k) + \mathbf{M}u(k-1) \in \Re^{nN_p} \quad (102)$$

allows to write

$$\boldsymbol{\Delta}\hat{\mathbf{u}} = \mathbf{K}(\mathbf{r} - \hat{\mathbf{x}}^{\mathbf{0}}) \quad (103)$$

Only the first line of $\boldsymbol{\Delta}\hat{\mathbf{u}}$ is needed, so the first line of $\mathbf{K}$ extracted, which called $\mathbf{k}^{\mathbf{T}}$.

$$\mathbf{K} = [\frac{\mathbf{k}^T}{\mathbf{k}''}] \text{ where } \mathbf{k}^T \in \Re^{m\times nN_p}$$

This means multiplying

$$\mathbf{k}^T = [\mathbf{I}_{m\times m} \mid \mathbf{0}_{m\times mN_c}]\mathbf{K}$$

The desired result is finally obtained

$$u(k) = u(k-1) + \mathbf{k^T}(\mathbf{r} - \hat{\mathbf{x}}^{\mathbf{0}}) \tag{104}$$

It can be seen how the integrator appears. The input vector $u(k)$ changes until $\mathbf{k^T}(\mathbf{r} - \hat{\mathbf{x}}^{\mathbf{0}})$ equals zero, or that $\mathbf{r} = \hat{\mathbf{x}}^{\mathbf{0}}$, which means zero steady-state error.

C.3 Predictive control algorithm for a model in the form $\hat{\mathbf{x}} = \mathbf{Z}x(k) + \mathbf{M}u(k-1) + \mathbf{F}\boldsymbol{\Delta}\hat{\mathbf{u}} + \hat{\mathbf{e}}$

The predictive control law for this model is now derived. There is a big similarity with a model of the form $\hat{\mathbf{x}} = \mathbf{Z}x(k) + \mathbf{M}u(k-1) + \mathbf{F}\boldsymbol{\Delta}\hat{\mathbf{u}}$, so that a similar reasoning can be made.

$$\boldsymbol{\Delta}\hat{\mathbf{u}} = (\mathbf{F^T Q F} + \boldsymbol{\Lambda})^{-\mathbf{1}}\mathbf{F^T Q}(\mathbf{r} - \mathbf{Z}x(k) - \mathbf{M}u(k-1) - \hat{\mathbf{e}})$$

Defining

$$\mathbf{K} = (\mathbf{F^T Q F} + \boldsymbol{\Lambda})^{-\mathbf{1}}\mathbf{F^T Q} \qquad \in \Re^{m(N_c+1)\times nN_p} \tag{105}$$

$$\hat{\mathbf{x}}^{\mathbf{0}} = \mathbf{Z}x(k) + \mathbf{M}u(k-1) + \hat{\mathbf{e}} \quad \in \Re^{nN_p} \tag{106}$$

$$\boldsymbol{\Delta}\hat{\mathbf{u}} = \mathbf{K}(\mathbf{r} - \hat{\mathbf{x}}^{\mathbf{0}})$$

Proceeding like in the previous section, and noting that the first line of $\mathbf{K}$ is again called $\mathbf{k^T}$, the desired result is found

$$u(k) = u(k-1) + \mathbf{k^T}(\mathbf{r} - \hat{\mathbf{x}}^{\mathbf{0}}) \tag{107}$$