ISSN 0280-5316
ISRN LUTFD2/TFRT--5735--SE

Control of an Inverted Pendulum (Pendubot)

Michael Kwapisz

Department of Automatic Control
Lund Institute of Technology
January 2005

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
January 2005

Document Number
ISRNLUTFD2/TFRT--5735--SE

Author(s)
Michael Kwapisz

Supervisor
Anders Rantzer LTH in Lund
K. Guemghar Lausanne in France

Sponsoring organization

Title and subtitle
Control of an Inverted Pendulum (Pendubot). (Reglering av en inverterad pendel (Pendubot))

Abstract
The aim of this project is to control a double linked inverted pendulum called the "Pendubot". A cascade control scheme has been developed at Laboratoire d'Automatique (EPFL) for this purpose. The structure of the cascade scheme is: Input-output feedback linearization and a linear controller in the inner-loop to assure the control of the *Á*-angle, the outer loop is responsible for stabilizing the *Ã*-angle. The whole idea is to make the inner loop much faster than the outer so to separate the system and to allow to control the *Á* and the *Ã* angle separately, This so called timescale separation is necessary for the cascade schemes e±ciency. This speed di®erence between the loops is accomplished by the linear controller. In the outer-loop three di®erent types of controllers will be implemented: a PD-structured controller, a non-linear controller and a Model Predictive controller (MPC).

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
58

Recipient's notes

Security classification

Contents

Chapter 1

Introduction

The "Pendubot" is a ABB Kreispendel that have been modified a bit. The purpose with this project was to implement a cascade controller on the "Pendubot" using a cascade scheme developed at Laboratoire d'Automatique (EPFL).

The cascade scheme has two main components: Input-output feedback linearization(IOFL) and a linear controller which form the so called inner-loop and the outer loop: PD-controller, non-linear controller or MPC.
A linear control is designed to control the ϕ-angle and to make the inner loop faster then the outer, this speed difference or this so called time-scale separation between the two loops is accomplished artificially by the large gains in the linear controller. The purpose of the outer-loop is to control the ψ-angle.

1.1 System description

The Pendubot is a robot arm mounted on a electrical motor on one point and a pendulum mounted at the opposite end. The pendulum, or the ϕ-angle is free and cannot be controlled directly, The electrical motor controls the arm or the ψ-angle and the its velocity $\dot{\psi}$. The output is ϕ and can only be controlled indirectly using ψ and $\dot{\psi}$. See 1.1 below.

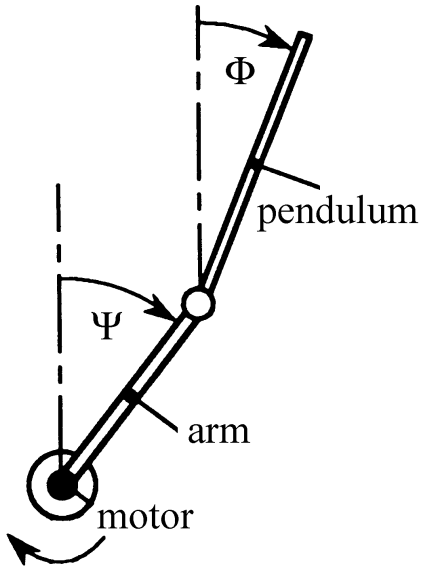

Figure 1.1: The Pendubot

Chapter 2

The Pendubot model

As in most case it is possible to describe a system mathematically, this is useful for the design of the controller, therefore the model accuracy is very important. The dynamics of the Pendubot can be described as follows:

$$\begin{aligned} J_1\ddot{\psi} + J\cos(\psi-\phi)\ddot{\phi} + J\sin(\psi-\phi)\dot{\phi}^2 - g_1\sin(\psi) + b_1\dot{\psi} + c_1 sign(\dot{\psi}) &= b\bar{u} \\ J_2\ddot{\phi} + J\cos(\psi-\phi)\ddot{\psi} - J\sin(\psi-\phi)\dot{\psi}^2 - g_2\sin(\phi) + c_2 sign(\dot{\phi}-\dot{\psi}) &= 0 \end{aligned} \tag{2.1}$$

where J_1, J_2 and J are the different inertias, $g_1 = m_{arm}\frac{l_{arm}}{2} + m_{pend}l_{arm}$ and $g_2 = m_{pend}\frac{l_{pend}}{2}$, where l_{arm} is the length of the arm and l_{pend} is the length of the pendulum. bu is the applied torque (Nm), where b is the gain between the input u (the voltage) and the torque. The constants are given in Table 2.1. c_1 and c_2 are the static friction coefficients and b_1 is the viscous friction coefficient.

From equation set 2.1 we can see that it is possible to eliminate the effects of friction directly in the first equation by applying following control law:

$$\bar{u} = \frac{1}{b}(u + b_1\dot{\psi} + c_1 sign(\dot{\psi})) \tag{2.2}$$

Unfortunately it is not possible to eliminate the friction effects in the second equation directly, hopefully this should not affect the performance significantly. Just by moving around the arm and pendulum we can feel that the friction force affecting the arm is a lot larger in magnitude than the one affecting the pendulum, therefore the friction term $c_2 sign(\dot{\phi}-\dot{\psi})$ is neglected.

J_1	0.22	$\mathrm{kg\,m^2}$
J_2	0.019	$\mathrm{kg\,m^2}$
J	0.0185	$\mathrm{kg\,m^2}$
g_1	1.7	Nm
g_2	0.36	Nm
b	3.2	Nm/V

Table 2.1: System parameters

Writing the equations in 2.1 in state-space form yields the following equations 2.3.

$$\begin{aligned} &\dot{x} = f(x) + g(x)b\bar{u}, \qquad x(0) = x_0 \\ &y = \phi \end{aligned} \tag{2.3}$$

where $x = [\ \phi\ \dot{\phi}\ \psi\ \dot{\psi}\ ]^T$ are the states, $\bar{u}$ the input, y the output , x_0 the initial conditions, and where f and g are the functions describing the system dynamics:

$$f = \begin{bmatrix} \dot{\phi} \\ -J_1\alpha - J\cos(\psi-\phi)\beta \\ \dot{\psi} \\ J\cos(\psi-\phi)\alpha + J_2\beta \end{bmatrix} \tag{2.4}$$

$$g = \begin{bmatrix} 0 \\ -\frac{J\cos(\psi-\phi)}{a} \\ 0 \\ \frac{J_2}{a} \end{bmatrix} \tag{2.5}$$

with:

$$\begin{aligned} a &= J_1J_2 - J^2\cos^2(\psi-\phi) \\ \alpha &= \frac{1}{a}(-g_2\sin\phi - J\sin(\psi-\phi)\dot{\psi}^2) \\ \beta &= \frac{1}{a}(g_1\sin\psi - J\sin(\psi-\phi)\dot{\phi}^2) \end{aligned}$$

Setting $x = 0$ and $u = 0$ in above equation yields $f(0) = 0$ since $\alpha(0) = 0$ and $\beta(0) = 0$, suggesting that this is an equilibrium point. Since the point [0 0] is the point were the "Pendubot" is pointing upwards, the pendulum is inverted, this intuitively suggests that the equilibrium point is unstable.

2.1 The zero-dynamics

The zero-dynamics are linked with the non-minimum phase behavior of the pendubot. Non-minimum phase means that the systems zeros are unstable. In the linear case a plant is non minimum phase when the zeros of the system are in the right half plane in the frequency domain, $s_{zeros} > 0$.
Investigating the zero-dynamics. Using equation set 2.1 and by setting $y = \phi = 0$ yields:

$$J_1\ddot{\psi} - g_1 sin(\psi) = u \quad (2.6)$$

$$Jcos(\psi)\ddot{\psi}^2 - Jsin(\psi)\dot{\psi}^2 = 0 \quad (2.7)$$

Simplifying further yields:

$$\ddot{\psi} = \frac{1}{J_1}(g_1 sin(\psi) + u) \quad (2.8)$$

$$\ddot{\psi}^2 = tan(\psi)\dot{\psi}^2 \quad (2.9)$$

According to equation 2.9 if $\psi = \xi$, where $|\xi| > 0$ is a very small number, then $|\ddot{\psi}| > 0$ this yields that the plant quickly diverges, hence the zero dynamics are unstable, the pendubot is therefore non-minimum phase. Please notice that the speed which the zero dynamics diverges is dependent on $\dot{\psi}$.

2.2 The relative degree

Another important feature is the systems relative degree. In the linear case the relative degree of a linear system with input u and output y is the difference between the number of poles and the number of zeros.

In the nonlinear case the relative degree is, the number of times the output has to be differentiated with respect to time until the input appears.

This property can be expressed as follows $\{ L_g L_f^{r-1} h(x) \neq 0,\ L_g L_f^i h(x) = 0, \forall i < r-1\ ,\ \forall x\ \}$, where $L_f N(x) = \frac{\partial N}{\partial x} f(x)$ is the Lie derivative of N and r is the relative degree.
Calculating the relative degree r:

$$L_g h(x) = 0 \tag{2.10}$$

$$L_g L_f h(x) = \frac{-J_2}{(-J_1 J_2 + J^2 cos(\psi - \phi)^2)} \neq 0 \tag{2.11}$$

Equation 2.10 and 2.11 suggest that the system has a relative degree $r = 2$.

Chapter 3

Control theory

This chapter will discuss the approach finding a controller for the Pendubot. As mentioned before the three main controllers will be described more thoroughly in this section.

3.1 The Input-Output Feedback Linearization (IOFL)

The IOFLs purpose is to by feedback making the system behave linear between input and output, if this is accomplished, linear control strategies can be applied to control the input-output behavior.
Knowing that the relative degree of the system is $r = 2$, yields following control law for the IOFL:

$$u = \frac{v - L_f^r h(x)}{L_g L_f h(x)} \tag{3.1}$$

where $L_f N(x) = \frac{\partial N}{\partial x} f(x)$ is the Lie derivative of N.

Linearizing the system through feedback consists of inverting the system dynamics. Since it's not possible to invert the system completely in our case, it is only possible to partially invert the system, therefore some residual dynamics called the internal dynamics remains. One important issue is that the IOFL does not stabilize the internal dynamics. Therefore an additional controller is needed.

3.1.1 The internal states and dynamics

After applying IOFL internal dynamics (IDs) remains. Using the non-linear transformation $z = T(x)$, where $z = [y\ \dot{y}\ \eta^T]$ and where η are the so called internal states (ISs), the IDs can be described as follows in a general expression:

$$\begin{aligned} \ddot{y} &= v \\ \dot{\eta} &= \mathcal{Q}(\eta, y, \dot{y}, v) \end{aligned} \tag{3.2}$$

The choice of the internal dynamics are free, the internal dynamics are chosen so that they are independent of the input v.
Assuming that the input-output behavior is much faster than the internal dynamics an assumption is made that the system is in a quasi-steady state. Also, we assume that: $y = \phi$ chases y_{ref} slowly therefore $\Delta y = \Delta y_{ref}$, $y = y_{ref}$ and $\dot{y} = \dot{y}_{ref}$, assuming this yields $\dot{\eta} = \mathcal{Q}(\eta\ ,\ y_{ref},\ 0,\ 0) = \mathcal{Q}(\eta\ ,\ y_{ref}))$, these are the so called reduced internal dynamics (RID).
The candidate for reduced internal states(RIS) is:

$$\begin{aligned} \eta_1 &= Jsin(\psi - \phi) \\ \eta_2 &= J\dot{\psi}cos(\psi - \phi) + J_2\dot{\phi} \end{aligned} \tag{3.3}$$

which yields following RIDs:

$$\begin{aligned} \dot{\eta}_1 &= \eta_2 \\ \dot{\eta}_2 &= g_2\lambda \end{aligned} \tag{3.4}$$

where $\lambda = sin(y_{ref})$, using this transformation yields completely linear RIDs. As seen above the RIDs behave as a double integrator and need to be stabilized. The stabilization of the RIDs is the reason why a cascade scheme is needed.

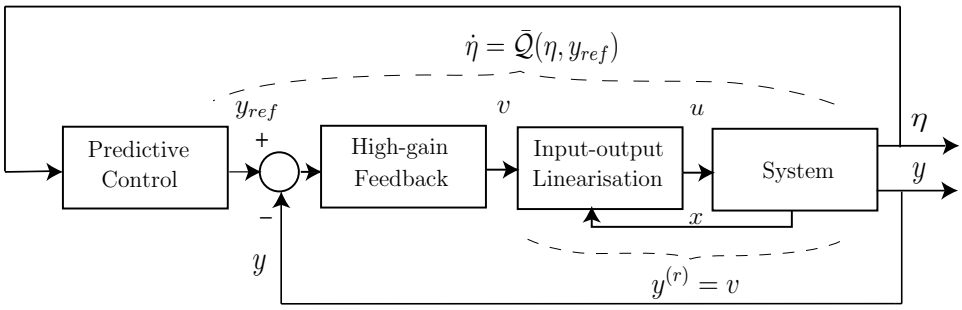

Figure 3.1: The cascade control scheme

3.2 The cascade scheme

The problem with unstable internal dynamics after applying IOFL motivates the use of a cascade scheme. The cascade scheme will contain of two loops, an inner loop and an outer loop.

The inner loop will contain an linear controller to control the output (ϕ-angle) while the outer loop will stabilize the internal dynamics (effectively the ψ-angle). A diagram of the cascade scheme is illustrated in figure 3.1.

In figure 3.1 it is written "predictive control", however other controllers can be put in at its place.
The cascade scheme controller is subject to two conditions. Firstly the inner loop have to be faster than the outer loop meaning that the bandwidth of the inner loop (ω_{inner}) has to be larger than the bandwidth of the outer loop (ω_{outer}). This yields the condition: $\omega_{inner} >> \omega_{outer}$.
This difference in bandwidth gives a time-scale separation between the inner- and outer-loop. Having a sufficient difference in bandwidths or time-scale separation allows control of the ϕ-angle separately from the ψ-angle.

Secondly, ideally the reference trajectory should be reached within a sub-sampling period $y(t_0 + h_{sub}) = y_{ref}$ for best performance. Where the reference, y_{ref}, is the desired output(ϕ).

3.3 The inner-loop

The inner-loop contains of the IOFL and a linear controller. The IOFL's purpose is to linearize the input-output behavior as mentioned before and the purpose of the linear controller is to control the output and to artificially create a time scale separation between the inner- and the outer-loop.

Since the IOFL yields a input-output linear system, it is possible to use simple linear design methods to control the output.
A PD-structured state-feedback control law is chosen:

$$v = -K_{\phi}(y - y_{ref}) - K_{\dot{\phi}}\dot{y} \tag{3.5}$$

Where K_{ϕ} and $K_{\dot{\phi}}$ is chosen by pole placement.

3.4 The outer-loop

The outer-loop will contain of several different types of controllers, the purpose of these controllers will however remain the same: *stabilize the internal dynamics*. This is done by the outer-loop controller which computes a ψ-angle stabilizing control input y_{ref} to be passed on as input to the linear controller in the inner-loop.

3.4.1 The PD-controller

The PD-controller represents the simplest type of controller in the outer-loop, using linearized internal states in the state-feedback control law.
Linearizing the internal states in equation 3.3 yields:

$$\begin{aligned} \eta_1^* &= J\psi \\ \eta_2^* &= J\dot{\psi} \end{aligned} \tag{3.6}$$

The outer-loop poles can there after be placed by applying a state-feedback control law:

$$y_{ref} = -K_{\eta_1^*}\eta_1^* - K_{\eta_2^*}\eta_2^* \tag{3.7}$$

3.4.2 The Non-Linear Controller (NLC)

The only difference between the PD-controller and the NLC is that the NLC uses the non-linear RISs as in equation 3.3 as states in the following

state-feedback control law:

$$\begin{aligned} \lambda &= -K_{\eta_1^*}\eta_1 - K_{\eta_2^*}\eta_2 \\ \lambda &= -K_{\eta_1^*}Jsin(\psi-\phi) - K_{\eta_2^*}(J\dot{\psi}\cos(\psi-\phi) + J_2\dot{\phi}) \end{aligned} \tag{3.8}$$

where $K_{\eta_1^*}$ and $K_{\eta_2^*}$ are the feedback gains chosen by pole placement just as for the PD-controller and $y_{ref} = \arcsin(\lambda)$.

3.4.3 The Model Predictive Controller (MPC)

The idea with the MPC is to simulate the system in the future, over a predefined prediction horizon T. An input signal λ is found to minimize the cost function J just as stated below in problem specification 3.9. The input is then applied over a sampling (or sub-sampling) period and then the whole procedure is repeated.

$$\begin{aligned} J(y_{ref}) &= \min_{\lambda([t,t+T])} \left\{ \frac{1}{2}\eta^T P\eta(t+T) + \right. \\ &\quad \left. \frac{1}{2}\int_t^{t+T} \eta^T Q\eta + R\,\lambda^2(\tau)d\tau) \right\} \end{aligned} \tag{3.9}$$

$$\begin{aligned} &subject\ to: \\ \dot{\eta}_1 &= \eta_2 \\ \dot{\eta}_2 &= g_2\lambda \\ \lambda &= sin(y_{ref}) \\ \\ \eta_1(t_0) &= \eta_1^0 \\ \eta_2(t_0) &= \eta_2^0 \end{aligned}$$

where, P, Q and R are weighting matrices, T is the prediction horizon, J is the cost function to be minimized and y_{ref} is the control input. Solving the optimization gives $\lambda^\star$ and $J^\star$ which is the optimal input and the minimum cost. The optimal control input is $y_{ref}^\star = arcsin(\lambda^\star)$. The problem is non-autonomous with fixed time T.

Solving the optimization problem analytically having an infinitive prediction horizon ($T = \infty$) gives an ordinary LQR-control law (see appendix) with static gains just as in the PD-controller and in the NLC. Assuming

a non infinitive prediction horizon an online optimization method of problem 3.9 is chosen to compute y_{ref}. The final point weighting matrix P can however be chosen in such a manner that:

$$\frac{1}{2}\int_{t+T}^{\infty} \eta^T Q \eta(\tau) \quad \approx \quad \frac{1}{2}\eta^T P \eta(t+T) \tag{3.10}$$

Hence it will be possible to compare the two different controllers.

Chapter 4

Hardware and software

To this point background theory have been discussed, the next step is to implement the theory to the actual system, for this we need hardwares and softwares to establish a connection between the controller and the physical plant. A hardware map is illustrated in figure 4.1. A more detailed explanation and specification about the hardware can be read reference [3].

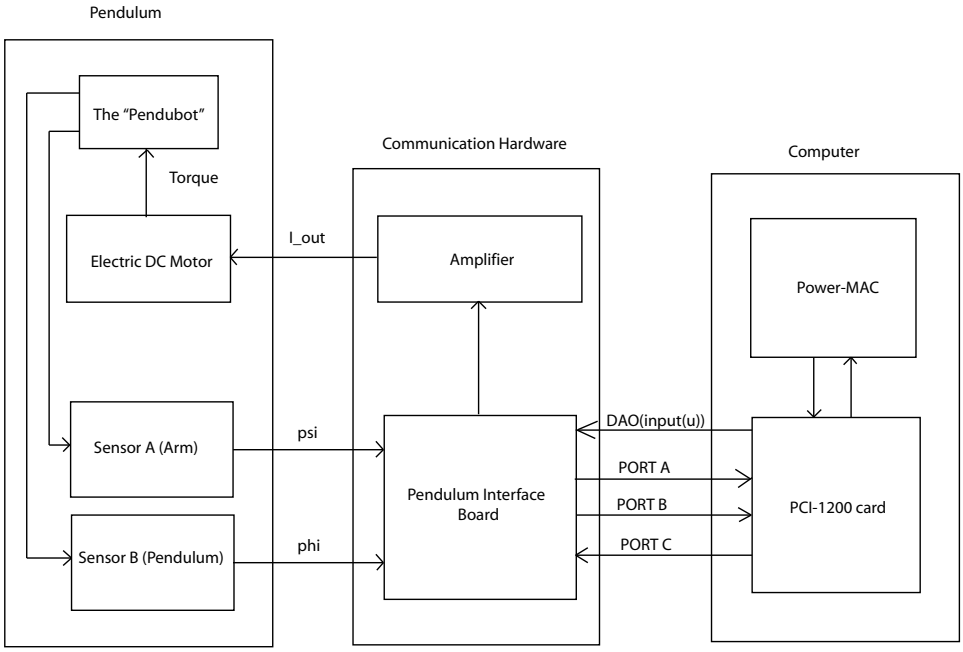

Figure 4.1: Hardware map

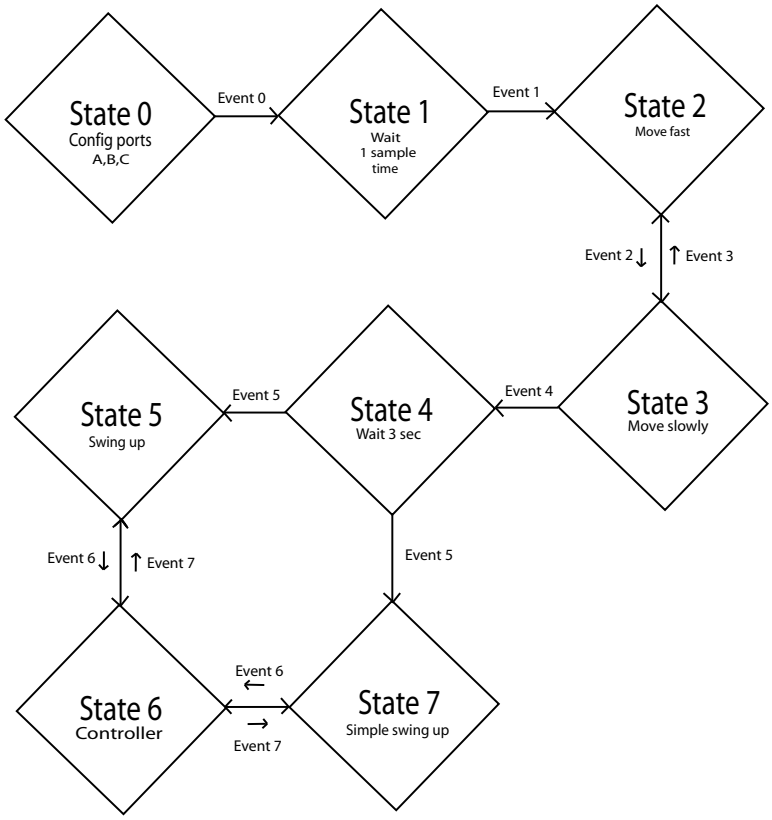

Figure 4.2: Real-time chain

The software used was: `LabView 6i`, `CodeWarrior 1.7.4` and `Matlab 5.2` for MAC.

`LabView 6i` provided the I/O between the plant and the controller(C-code), graphical interface and the real-time kernel.
Filename: `CascadeContWithMPC2.GUI`.

`Codewarrior 1.7.4` was the C-compiler.
Filename: `Task1.c`.

`Matlab 5.2` was used for calculations and simulations.
The map of the real-time program is illustrated in 4.2 State explanation:

- **State 0** : Configuration of the I/O ports A, B and C.

- **State 1**: Wait one sample time, this is necessary when calculating the angular velocity, in State 0 the first measurement is not available due to the fact that the I/O ports are not defined yet.
- **State 2**: Move fast clockwise. The speed of the arm is controlled by a PI-controller.
- **State 3**: Move slowly clockwise. The speed is PI-controlled as in State 2, but slower. State 2 and 3 is used for calibrations of the sensors.
- **State 4**: Wait 3 seconds. Manually reseting the sensors here.
- **State 5**: Swing up using IOFL with ψ as output and destabilize ϕ.
- **State 6**: Controllers, outer-loop controllers.
- **State 7**: Simple swing up, swing arm back and forth until the controller catches it.

Event explaination:

- **Event 0** : Next sample period.
- **Event 1**: Next sample period.
- **Event 2**: Arm pointing almost straight down.
- **Event 3**: Arm pointing passed: arm pointing down-area,
- **Event 4**: Arm pointing downwards pendulums angular velocity is zero, $\dot{\phi} = 0$.
- **Event 5**: After 3 sec go to swing up of choice.
- **Event 6**: ψ is in upper half($|\psi| < \pi/2$) and $|\phi| < \pi/4$, also for simple swing up $|\dot{\phi}| < 2.5$.
- **Event 7**: $|\psi| > \pi/2$.

Chapter 5

Implementing the cascade controller

Implementing the cascade controller scheme on the actual system was pretty much straight forward even though a lot of problems had to be solved on the way.

The actual core program, the real-time kernel where the controller was to be implemented already existed from previous work but needed to be modified. The initialization process defining the I/O ports, the basic commands i.e reading the sensors and converting the sensor values to radians, had already been implemented. The basic commands are:

`short GetA(void)`
Reading sensor A, ranging (-8192 . . +8191), 0 when arm is pointing completely downwards. Corresponding to the ψ angle.

`short GetB(void)`
Reading sensor B, ranging (-8192 . . +8191), 0 when the angle between the arm and pendulum is 0 (arm and pendulum are parallel and fully extended).

`void DAout(double u)`
Converting and applying the control signal u. u ranging (-5 . . +5V) is converted to binary values ranging (-2048. . 2047). The converted u is sent to the PCI card by the command Dabin.

`void getValues(double *psi, double *psid, double *phi, double *phid, double h)`
Calculating the angles ψ and ϕ in radians as in figure 1.1 as well as filtering

the angles to obtain the angle velocities $\dot{\psi}$ and $\dot{\phi}$. The input is h (sampling time), the outputs are psi(ψ), psid($\dot{\psi}$), phi(ϕ) and phid($\dot{\phi}$).

5.1 Implementing the inner loop controller - IOFL and linear controller

The main problems that arose implementing this part were: friction, noise and actuator limitation.
Without proper friction compensation the controller performed rather poorly, giving limit cycles. No compensation yielded the system unstable.

Noise caused problems when calculating the angular velocities causing a very 'nervous' control signal.

The limitation on the actuator caused that the time-scale separation wasn't large enough, yielded decreased tracking performance of the reference signal.
The function for IOFL and linear controller in Task1.c:

```
d* FeedBackLinController(d* phi, d* phid, d* psi, d* psid, d* phi_ref, d* t)
```

where d* means `double` format: Here are the angles $phi = \phi$ and $psi = \psi$. The angular velocities $phid = \dot{\phi}$ and $psid = \dot{\psi}$. The input $phi\_ref = y_{ref}$ is the reference angle. t is the sampling time.

5.1.1 Implementation of the linear controller

The feedback gains of the linear controller in equation 3.5 was chosen to be: $K_\phi = \frac{1}{\epsilon^2}$ and $K_{\dot{\phi}} = \frac{2}{\epsilon}$ yielding the control law:

$$v = -\frac{1}{\epsilon^2}(y - y_{ref}) - \frac{2}{\epsilon}\dot{y} \tag{5.1}$$

which placed the two inner-loop poles at $1/\epsilon$. Hence only one parameter was tuned to place the poles.
As mentioned previously our objective is to have the fastest inner loop as possible, to guarantee a time scale separation, this wish have however limitations, in our case $\epsilon_{min} \approx 0.05(s/rad)$, placing the poles of the inner loop at $s = -20(rad/s)$. A smaller ϵ made the arm vibrate heavily, a sign of a limitation. The consequence of choosing $\epsilon > 0.05$ was that the MPCs performance decreased. Using a higher ϵ worked however for the NLC and the PD-controller.

5.1.2 Friction

In our case the frictions were assumed to be static and viscous. The constants b_1 and c_1 from equation 2.1 were experimentally determined by basic trail and error method. Table 5.1 below contains the friction constants found to work best:

b_1	0.5	Nm/rads^{-1}
c_1	2.41	Nm

Table 5.1: Friction parameters

These values did not correspond to previous paper [3], this might suggest that the frictions could have changed over time. Changed friction values over short time was also noticed since constant tuning was required. Difference could be noticed after the Pendubot had been running for a while, then the friction seemed to decrease in magnitude.

5.1.3 Noise and actuator limitation

As always when dealing with sensors they are subject to noise, this noise can decrease the performance of the system gravely if not taken care of. A way to limit the effects of the noise is to filter the signals, this however introduces a delay in the feedback loop. A filter with a well defined cut off frequency is of a higher order yielding greater delay while a filter with a flatter cut off frequency doesn't filter as good. Designing a filter is therefore a trade off problem.

The choice of filter for the Pendubot was a digital Butterworth filter of order 3 which gave satisfying filtering performance while not delaying the feedback-loop too much. The choice of cut off frequency was so that the filter would be at least 5 times faster than the inner loop so that the filter dynamics doesn't affect the actual system dynamics.
This filter specification yielded following cut off frequency:

$$f_{cut} = \frac{5}{2\pi\epsilon}\ (Hz) \tag{5.2}$$

where $\frac{1}{\epsilon}$ are the inner loop poles, that means that the filter design is dependent on the inner loop bandwidth and hence on the actuator's limitations. According to equation 5.2 above, the cut off frequency was chosen to be

$f_{cut} = 20Hz$.

Another limitation was the bounds on the input signal (-5V . . 5V), meaning that the input quickly got saturated and hence the system was not controllable. This input saturation affected the performance close to the singularity area of the IOFL where we need infinitive control input to maintain control, this inevitably leads to loss of control.

5.2 Implementing the outer-loop - the PD-controller

The first PD-controller was implemented for the verification of the IOFL, since the ψ angle isn't controlled using IOFL + linear controller, but ϕ is, hence $\phi \approx 0$. When not controlling the ψ-angle the ϕ-angle quickly diverged and when it reaches the singularity point $cos(\psi - \phi) = 0$ or $cos(\psi) \approx 0$ when $\psi = \pm\pi/2$ the control signal saturated and the controllability was lost. This could be prevented by implementing the PD-controller on ψ in the outer loop, with the purpose of controlling $\psi \to 0$ by generating a reference trajectory that was passed to the IOFL and linear controller, hence keeping the Pendubot in the controllable zone $|\psi| < \pi/2$.

Firstly, the tuning of the gains was trial and error just to stabilize ψ. After verifying that the IOFL + linear controller worked properly the gains of the PD was revised, this time to satisfy the performance specification of good tracking of the reference trajectory and relative quick convergence of $\psi \to 0$. Even though ϵ can be as low as $\epsilon = 0.05$ an $\epsilon = 0.1$ was chosen because of it's smoother control input, this placed the inner loop poles at $s = -10$ (rad/s). As a rule of thumb the outer-loop was designed to be at least 5 times slower to separate the two systems, making them independent of each other. This rule of thumb placed the poles in the range $0 > s > -2$. The feedback gains were then calculated to place the linearized internal dynamics poles at the specified location.

By trying different options of pole placement, the one considered satisfying the specifications best was then picked. The different option are illustrated below:

for best overall performance the poles was chosen to be $[-1 - 1.5]$ for $\epsilon = 0.1$ and [-1 -1] for $\epsilon = 0.05$. These poles will later work as a guideline when designing the NLC and the MPC. in C-code y_{ref} was expressed accordingly (poles at [-1 -1.5]):

```
yref = -0.0771*psi - 0.1285*psid
```

ϵ	Poles	Convergence	Tracking	disturbance rejection/recovery
0.1	[-1 -1]	too slow	good	good
0.1	[-2 -2]	fast	bad , y_{ref} too aggressive	bad, very sensitive
0.1	[-1.5 -1.5]	fast	poor , y_{ref} quite aggressive	not so good, sensitive
0.1	[-1 -1.5]	fast	acceptable	quite robust, quite good recovery from disturbances
0.05	[-1-1]	fast	good	acceptable
0.05	[-2 -2]	overshoots	poor, very aggressive	very poor/unstable, oscillating when disturbed lightly
0.05	[-1.5 -1.5]	overshoots	not so good	poor, oscillating when disturbed
0.05	[-1 -1.5]	fast	acceptable	sensitive to disturbance

Table 5.2: Choosing inner loop poles - performance table

5.3 Implementing the outer-loop - The NLC

The only difference between the PD-controller and NLC is that in the PD-controller the internal dynamics are linearized, in the NLC the internal dynamic are just as stated in 3.3 giving the control law as in 3.8.
Note that $y_{ref} = arcsin(\lambda)$:

$$\lambda = -K_{\eta_1^*} J sin(\psi - \phi) - K_{\eta_2^*}(J\dot{\psi}\cos(\psi - \phi) + \underline{J_2\dot{\phi}}) \qquad (5.3)$$

In the actual implementation the underlined part was neglected because of phid's ($\dot{\phi}$) sensitivity to noise. Neglecting this part yielded a smother control signal without loss in performance.

The big advantage over the linear PD is that the NLC act less aggressive when ψ is far from 0, therefore giving smoother control input and a

calmer behavior hence only the NLC will be discussed from now on. This can be explained as follows: Rewriting the control law 5.4 after neglecting the phid-part gives as follows:

$$\lambda = -K_{\eta_1^*} J sin(\psi) - K_{\eta_2^*} J\dot{\psi}\cos(\psi) \tag{5.4}$$

Rewriting once more by introducing the new non-static, non-linear gains $K_\psi(\psi,\phi)$ and $K_{\dot{\psi}}(\psi,\phi)$:

$$\lambda = -K_\psi(\psi,\phi)\psi - K_{\dot{\psi}}(\psi,\phi)\dot{\psi} \tag{5.5}$$

where:

$$\begin{aligned} K_\psi(\psi,\phi) &= K_{\eta_1^*} J\frac{sin(\psi-\phi)}{\psi} \\ K_{\dot{\psi}}(\psi,\phi) &= K_{\eta_2^*} J cos(\psi-\phi) \end{aligned} \tag{5.6}$$

Revealing the state feedback structure with dynamic gains. Note that:

$$\lim_{\psi,\phi\to 0}\frac{sin(\psi-\phi)}{\psi} = 1. \tag{5.7}$$

Since the ϕ is controlled to zero faster than ψ we can assume that $\phi \approx 0$ and to simplify, let say $sin(\psi) \approx \psi$. this leaves the control law:

$$\begin{aligned} K_\psi(\psi,\phi) &= K_{\eta_1^*} J \\ K_{\dot{\psi}}(\psi,\phi) &= K_{\eta_2^*} J cos(\psi) \end{aligned} \tag{5.8}$$

Above equation shows that for larger ψ:s the gain on the angular velocity psid ($\dot{\psi}$) is decreased, hence resulting in smoother control signal for larger ψ-angles.

In C-code the NLC control law is written as follows (poles at [-1 -1.5]):

```
yref = asin(-0.0771*sin(psi-phi) - 0.1285*psid*cos(psi-phi))
```

Note that the phid ($\dot{\phi}$) part is missing. `yref` is passed on to the command `FeedBackLinController` as `phi_ref`. The feedback gains are for $\epsilon = 0.1$. The feedback gains on the internal dynamics are the gains illustrated above divided by the inertia J, K_i/J.

5.3.1 NLC under sub-sampling

If the MPCs wouldn't finish the optimization within a sampling period (5 ms), then sub-sampling was useful, allowing the optimization more time to finish. A NLC under sub-sampling was therefore introduced so that the NLC and MPCs could be compared. `subNLC` is the C-function name for the NLC subject to sub-sampling.

5.4 Implementing the outer-loop - The MPCs

Several issues arose when implementing the MPCs, numerical optimization and integration methods had to be chosen, parameters of the optimization methods had to be tuned. Therefore several MPCs were implemented and the ones that seemed the most appropriate were chosen.

5.4.1 Optimization method

Solving problem 3.9 with a fixed non-infinitive prediction horizon was done by an online solver. The solver of choice was the so called "Quasi-Newton method".

In Newton's method, $J(\lambda^{(k)+\delta}) \approx q^{(k)}(\delta) = J^k + g^{(k)^T}\delta + \frac{1}{2}\delta^T G^{(k)}\delta$, where $\delta = \lambda - \lambda^{(k)}$ and $q^{(k)}(\delta)$ is the resulting quadratic approximation for iteration k. then the iterate $\lambda^{(k+1)} = \lambda^{(k)} + \delta^{(k)}$, where the correction minimizes $q^{(k)}(\delta)$. This method requires zero, first and second derivatives.

The Quasi-Newton method on the other hand only requires the zero and first derivative. The Quasi-Newton method works like Newton's method with line search, the only exception that $\mathbf{G^{(k)-1}}$ is approximated by using a symmetric positive definite matrix $\mathbf{H^{(k)}}$, which is corrected or updated after every iteration, hence the second derivative doesn't need to be calculated analytically.
The Quasi-Newton is described in pseudo-code below:
-Initialize algorithm-
step 0: H is initialized as I and initial conditions are set, a system integration takes place to estimate what the initial conditions will be when the input (λ) will be applied

-Algorithm-
step 1: Initial conditions are set (have to be done before every iteration)

step 2: run quasi-newton algorithm
(a) -integrate system- subject to $\lambda(\mathbf{k})$
(b) $\mathbf{s}(\mathbf{k}) = -\mathbf{H}(\mathbf{k})\mathbf{g}(\mathbf{k})$
(c) line search along $\mathbf{s}(\mathbf{k})$ *giving* $\lambda(\mathbf{k}+\mathbf{1}) = \lambda(\mathbf{k}) + \alpha(\mathbf{k})\mathbf{s}(\mathbf{k})$ *(α is the tuning parameter usually set to 1)*
(d) calculate $\delta(\mathbf{k}) = \alpha(\mathbf{k})\mathbf{s}(\mathbf{k}) = \lambda(\mathbf{k}+\mathbf{1}) - \lambda(\mathbf{k})$
(e) -integrate system- subject to $\lambda(\mathbf{k}+\mathbf{1})$
(f) calculate $\gamma(\mathbf{k}) = \mathbf{g}(\mathbf{k}+\mathbf{1}) - \mathbf{g}(\mathbf{k})$
(g) update $\mathbf{H}(\mathbf{k})$ *giving* $\mathbf{H}(\mathbf{k}+\mathbf{1})$ *using:*

$$\mathbf{H}(\mathbf{k}+\mathbf{1}) = \mathbf{H}(\mathbf{k}) + \frac{(\delta(\mathbf{k}) - \mathbf{H}(\mathbf{k})\gamma(\mathbf{k}))(\delta(\mathbf{k}) - \mathbf{H}(\mathbf{k})\gamma(\mathbf{k}))^{\mathbf{T}}}{(\delta(\mathbf{k}) - \mathbf{H}(\mathbf{k})\gamma(\mathbf{k}))^{\mathbf{T}}\gamma(\mathbf{k})} \tag{5.9}$$

Integrating forward the system using Euler integration method from $t = t_0$ to $t = T$, where T is the prediction horizon, gives as follows:
For-loop from $k = 0$ to $k = T/h$, where h is the sampling period. Integrating forward the reduced internal dynamics:

$$\begin{aligned} \eta_1(k+1) &= \eta_1(k) + h\eta_2(k) \\ \eta_2(k+1) &= \eta_2(k) + hg_2\lambda(k) \end{aligned}$$

Integrating the sensitivities

$$\begin{aligned} \frac{\partial\eta_1}{\partial\lambda}(k+1) &= \frac{\partial\eta_1}{\partial\lambda}(k) + h\frac{\partial\eta_2}{\partial\lambda}(k) \\ \frac{\partial\eta_2}{\partial\lambda}(k+1) &= \frac{\partial\eta_2}{\partial\lambda}(k) + hg_2 \end{aligned}$$

Integrating the gradient g:

$$\begin{aligned} g(k+1) = {} & g(k) + h(q_{11}\eta_1(k+1)\frac{\partial\eta_1}{\partial\lambda}(k+1) + \\ & (q_{12}+q_{21})(\eta_1(k+1)\frac{\partial\eta_2}{\partial\lambda}(k+1) + \eta_2(k+1)\frac{\partial\eta_1}{\partial\lambda}(k+1)) + \\ & q_{22}\eta_2(k+1)\frac{\partial\eta_2}{\partial\lambda}(k+1) + R\lambda(k)) \end{aligned}$$

End for-loop. Penalizing the final position.

$$\begin{aligned} g(N) &= g(N) + (p_{11}\eta_1(N)\frac{\partial \eta_1}{\partial \lambda}(N) + \\ &\quad (p_{12}+p_{21})(\eta_1(N)\frac{\partial \eta_2}{\partial \lambda}(N) + \eta_2(N)\frac{\partial \eta_1}{\partial \lambda}(N)) + \\ &\quad q_{22}\eta_2(N)\frac{\partial \eta_2}{\partial \lambda}(N) + R\lambda(N-1)) \end{aligned}$$

where P,Q and R are the weighting matrices from 3.9 and N is the final time:

$$P = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}, \quad Q = \begin{bmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{bmatrix}, \quad R = R \tag{5.10}$$

As seen above the sensitivities $\frac{dx}{du}$ are computed and integrated instead of integrating the system with an input $\lambda + \Delta\lambda$ and λ

$$g(\lambda) = \frac{J(\lambda + \Delta\lambda) + J(\lambda)}{\Delta\lambda} \tag{5.11}$$

Using the sensitivities proved to be less time demanding and therefore chosen.

The dimension of λ was chosen to be 1 or 2. According to simulations a 2-dimensional λ performed better then a 1-dimensional λ. A 3-dimensional λ was never considered since it was too time demanding. A 1-dimensional λ gives a H of dimension 1 and g of dimension 1. Having a 2-dimensional λ yields a 2x2 H and a 1x2 g.
When using a 2-dimensional λ, λ_1 is applied in the simulated system from $t_0 \rightarrow \frac{T}{2}$ while λ_2 is applied from $\frac{T}{2} + h \rightarrow T$. However λ_2 will never be applied on the actual system.

5.4.2 Integration method

Four integration methods were considered, two of these were rejected before even implemented. RK45 was rejected because it was too complicated to implement together with fixed step sized Euler's method, the optimization integration uses Euler's method. RK2 was rejected because it didn't increase the integration precision enough.

1. Euler, fixed step size, order 1, (Euler)

2. Runge-Kutta, fixed step size, order 2, (RK2).

3. Runge-Kutta, fixed step size, order 4, (RK4).

4. Runge-Kutta, adaptive step size, order 4, (RK45).

Euler's method proved to be sufficient since the RIDs were quite simple in nature, a double integrator. RK4 was tested but the improvement in precision didn't affect the performance. However the higher precision in RK4 was needed when a MPC using non-linear internal dynamics was implemented (see appendix). RK4 was also slower than Euler's method. Execution time comparison:

- RK4, took 2.7ms for 10 optimization iterations.
- Euler,took 1.6ms for 10 optimization iterations

5.4.3 The MPC parameters

α, the first parameter to be set, since the Quasi-Newton method have a theoretical ability to converge after only $2N + 1$ iterations when having the right α, where N is the order of the system. $\alpha = 1$ worked best and was therefore chosen when λ was of 2-dimensions. When λ was of 1-dimension an $\alpha = 0.5$ was chosen, a smaller α:s yielded that the Quasi-Newton optimization converged slower but on the other hand it was more stable.
The prediction horizon, was the second parameter to be set. If we would simulate the whole system then the prediction horizon must be longer than the "dip" caused by the non-minimum phase feature of the system. Since our set of RIDs is only a double integrator, no restriction of how short the prediction horizon exists.
Since λ have the dimension $N_\lambda = 1$ or 2: $h_{sub} \leq \frac{T_{pred}}{N_\lambda}$. Leading to the choice of prediction horizon:

$$T_{pred} \geq N_\lambda \cdot h_{sub} \tag{5.12}$$

This condition states that the prediction horizon have to be longer than time the reference input y_{ref} acts on the system.

Other factors affecting the choice of prediction horizon is that the reference have to be reached within a sub-sampling period i.e $y(t_0 + h_{sub}) \approx y_{ref}$ and ideally also that $\dot{y}(t_0 + h_{sub}) \approx 0$.
Calculating the settling time for condition $y(t_0 + h_{sub}) \approx y_{ref}$. Firstly we

need an expression for the inner-loop transfer function which determines the settling time:

$$H(s) = \frac{\frac{1}{\epsilon^2}}{s^2 + \frac{2}{\epsilon}s + \frac{1}{\epsilon^2}} \tag{5.13}$$

To be able to calculate the settling time we need the systems step response. This is done by subjecting the system to a step input, i.e $\Theta(t) \xrightarrow{\mathcal{L}} \frac{1}{s}$ and transforming the resulting transferfunction*step-input back to the time-plane:

$$H(s)\frac{1}{s} \xrightarrow{\mathcal{L}^{-1}} -(1 + \frac{t}{\epsilon})e^{(-t/\epsilon)} + 1 \tag{5.14}$$

Solving in the time-plane the settling time (98%):

$$-(1 + \frac{T_s}{\epsilon})e^{(-T_s/\epsilon)} + 1 = 0.98 \tag{5.15}$$

Gives that $T_s \approx 5.8339\epsilon$, this yields the expression: $T_{pred} \geq N \cdot 5.8339\epsilon$. Equation 5.15 have to be solved numerically. This condition is not necessary however, the MPC will work even if this condition is not satisfied.

The feedforward gain If the condition,$y(t_0 + h_{sub}) \approx y_{ref}$ isn't fulfilled, what happens then?, Actually there is a way to maintain the MPC performance even if the condition is not satisfied, the trick is to apply a feedforward gain on y_{ref}, to decrease the error, $error = y_{ref} - y(t_0 + h_{sub})$. Using equation 5.14 the feedforward gain (ffgain) is calculated accordingly:

$$ffgain = \frac{1}{-(1 + \frac{h_{sub}}{\epsilon})e^{(-h_{sub}/\epsilon)} + 1} \geq 1 \tag{5.16}$$

This ffgain can also be tuned by trail and error, Table 5.3 should work as a guideline when implementing a feedforward gain. Simulated result using $\epsilon = 0.05$, $h = 5ms(sec)$, $h_{sub} = 0.2(sec)$ with $ffgain = 1.1008$ 5.1):
Practically the ffgain was only implemented on the MPC-2 for short subsampling periods hence it can be seen as an Ad-hoc solution.

5.4.4 Weighting matrices and feedback gains

Choosing the weighting matrices in optimization problem 3.9, was quite tricky when fixing the poles and calculating backwards to get Q, P and R.

ϵ	$h_{sub}(sec)$	T_{pred}	$ffgain$
0.1	0.2	$0.8 \geq 0.2$	1.6835
0.1	0.3	$0.8 \geq 0.6$	1.2487
0.1	0.4	$0.8 \geq 0.8$	1.1008
0.05	0.2	$0.8 \geq 0.4$	1.1008
0.05	0.3	$0.8 \geq 0.6$	1.0177
0.05	0.4	$0.8 \geq 0.8$	1.0030

Table 5.3: Examples of ffgains

h_{sub}	s-poles	s-feedback gains	z-poles	z-feedback gains
0.2	$[-1 \quad -1.5]$	[4.17 6.94]	[0.82 0.74]	[3.26 5.79]
0.3	$[-1 \quad -1.5]$	[4.17 6.94]	[0.74 0.64]	[2.90 5.32]
0.4	$[-1 \quad -1.5]$	[4.17 6.94]	[0.67 0.55]	[2.58 4.91]
0.2	$[-1 \quad -1]$	[2.78 5.56]	[0.82 0.82]	[2.28 4.81]
0.3	$[-1 \quad -1]$	[2.78 5.56]	[0.74 0.74]	[2.07 4.49]
0.4	$[-1 \quad -1]$	[2.78 5.56]	[0.67 0.67]	[1.89 4.20]

Table 5.4: Examples of when feedback-gains changes after discretizing

This was simple calculations for a continuos time system but when using a sub-sampling time h_{sub} which is $0.2 \rightarrow 0.8$ seconds it's hard to motivate that we still can consider the system as continuos.

According to the rule of thumb choosing sampling frequency $\omega_n h = 0.2 \rightarrow 0.6$ since we know that $h = h_{sub}$ and $\omega_n \sim 1$, this from pole-placement in the section discussing the PD-controller, we can see that the system should be considered as a discrete time system and therefore converted. Below is a table (Table 5.4) of continuos time poles and feedback gains (denoted as s-) and corresponding discrete time poles and feedback gains (denoted as z-) when sub-sampling at a period h_{sub}. All are for $\epsilon = 0.05$. the system as in equation 3.4.

As seen in the table above the feedback gains are changing when discretizing and because we are interested in the weighting matrices the idea of have the poles fixed and calculate the weighting matrices was discontinued instead feedback gains were calculated using matlab's `LQR`, the advantage in our case is using the same weighting matrices in a continuous `LQR` and in a discrete `LQR` (`dLQR`) will produce equivalent poles, hence it will make the calculations and comparisons easier.

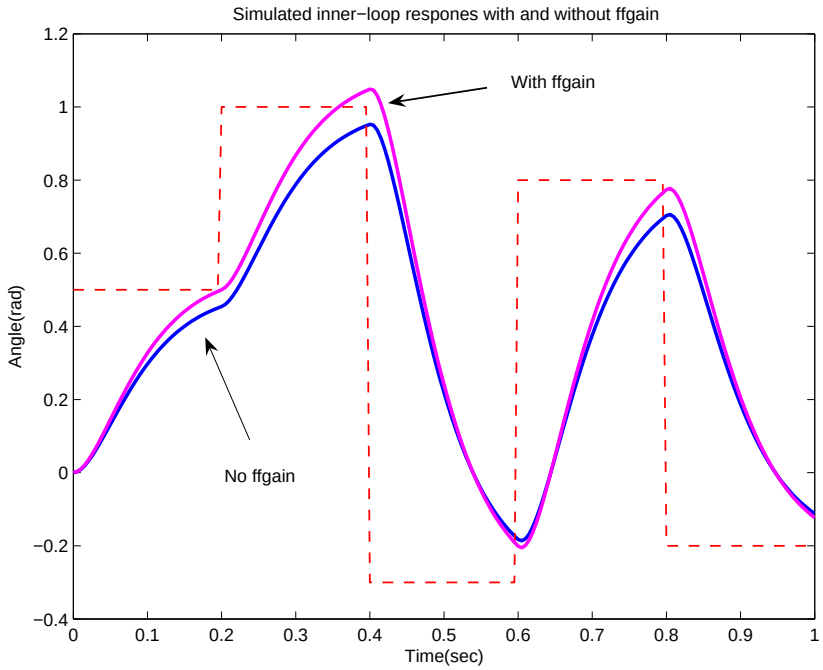

Figure 5.1: Comparing MPCs with and without feedforward gain, ffgain.

The poles in the outer loop have constraints however, firstly the outer loop have to be at least 5 times slower than the inner loop so that the dynamic will not not interfere with each other, this gives us the first condition.

$$poles \leq \frac{1}{5\epsilon} \tag{5.17}$$

Since $\epsilon = 0.05$, $poles \leq 4(rad/s)$.
A second condition which comes from Nyquist's theorem when discretizing continuous systems:

$$\omega_n h_{sub} = 0.2 \rightarrow 0.6 \tag{5.18}$$

where ω is the outer loops bandwidth. Below is a table of how ω_{nmax} varies depending on h_{sub}, ω_{nmin} have to be chosen in such manner so that the pendubot remain in the area $|\psi| < |\pi/2|$.

h_{sub}(sec)	ω_{nmax}
0.2	$1 \rightarrow 3$
0.3	$0.67 \rightarrow 2$
0.4	$0.5 \rightarrow 1.5$
0.5	$0.4 \rightarrow 1.2$
0.6	$0.33 \rightarrow 1$

Table 5.5: Table of were we can expect to encounter ω_{nmax} when subsampling

For the actual NLC and MPC design several control laws, Table 5.6, with different weighting matrices were calculated. These were later used if they fulfilled above mentioned conditions, R is always $R = 1$:

To be able to use table 5.6 controllers on the NLC , the feedback gains have to be calculated for the specific h_{sub}, this is done quickly in `Matlab` using:

```
Kz = dlqr(Az,Bz,Q,R)
```

where Az and Bz are the discretized system matrices using sampling time h_{sub}.

```
[Az, Bz, Cz, Dz] = ssdata(c2d(ss(A,B,C,D),hsub))
```

5.4.5 The resulting MPCs

Two main MPCs were implemented, MPC-1 and MPC-2. MPC-1 uses a 1-dimensional λ and MPC-2 uses 2-dimensional λ, because of good results when simulating the system with this dimension.

The MPCs uses the same optimization method - the Quasi-Newton method, with integrations of the sensitivities. MPC-1 will therefore have a H of dimension 1x1 and a g of dimension 1x1. MPC-2 however will use a H of dimension 2x2 and a g of dimension 1x2, hence more complex. the convergence parameter α was chosen $\alpha = 1$ for MPC-2 and $\alpha = 0.5$ for MPC-1.

The integration method is Euler's method through out for both system integrations and optimization integrations. Using Euler's method was suffi-

Controller :	Q-matrix	P-matrix	poles (in s)
(1)	$\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$	$\begin{bmatrix} 1.49 & 1.96 \\ 1.96 & 5.85 \end{bmatrix}$	$[-0.38 \pm 0.33i]$
(2)	$\begin{bmatrix} 1.0 & 0 \\ 0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 2.56 & 2.78 \\ 2.78 & 7.11 \end{bmatrix}$	$[-0.46 \pm 0.38i]$
(3)	$\begin{bmatrix} 2.0 & 0 \\ 0 & 2.0 \end{bmatrix}$	$\begin{bmatrix} 4.44 & 3.93 \\ 3.93 & 8.72 \end{bmatrix}$	$[-0.57 \pm 0.44i]$
(4)	$\begin{bmatrix} 5.0 & 0 \\ 0 & 5.0 \end{bmatrix}$	$\begin{bmatrix} 9.33 & 6.21 \\ 6.21 & 11.6 \end{bmatrix}$	$[-0.75 \pm 0.49i]$
(5)	$\begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$	$\begin{bmatrix} 16.6 & 8.79 \\ 8.79 & 14.6 \end{bmatrix}$	$[-0.95 \pm 0.50i]$
(6)	$\begin{bmatrix} 25 & 0 \\ 0 & 25 \end{bmatrix}$	$\begin{bmatrix} 36.3 & 13.9 \\ 13.9 & 20.2 \end{bmatrix}$	$[-1.31 \pm 0.30i]$
(7)	$\begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix}$	$\begin{bmatrix} 66.8 & 19.6 \\ 19.6 & 26.2 \end{bmatrix}$	$[-1.11\ ,\ -2.29]$
(8)	$\begin{bmatrix} 75 & 0 \\ 0 & 75 \end{bmatrix}$	$\begin{bmatrix} 96.1 & 24.1 \\ 24.1 & 30.8 \end{bmatrix}$	$[-1.06\ ,\ -2.93]$
(9)	$\begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$	$\begin{bmatrix} 125 & 27.8 \\ 27.8 & 34.6 \end{bmatrix}$	$[-1.05\ ,\ -3.45]$

Table 5.6: Controllers candidates

ciently accurate for the double integrator system as well as it was fast.

The prediction horizon was chosen to $T = h_{sub}$ for MPC-1 and $T = 2 \cdot h_{sub}$ for MPC-2.

The feedforward gain was only implemented on MPC-2.
In the actual C-program MPC-1 and MPC-2 are named `MPC_1` respectively `MPC_2`.

Chapter 6

Results and conclusions

In this chapter results and issues related to performance and limitations of the different controllers will be discussed.

Since there exist numerous different controller candidates, illustrated in Table 5.6, and since there also exists numerous of sub-sampling period choices only a few candidates for comparison will be picked and plotted.

Firstly, no sub-sampling was chosen, the controllers tested were NLC and MPC-1. The minimal sub-sampling period for MPC-2 was $h_{sub} = 0.025$ (sec), which is 5 times longer than h. Choosing a prediction horizon of 0.5 sec made it possible to do 175 optimization iterations per sub-sampling period. Choosing a shorter prediction than 0.5 wasn't possible due to bad results for the MPC-2. After this try, sub-sampling periods of 0.2, 0.4 and 0.6 seconds were tried.

The disturbance applied was a simple 'touch' with the finger on the pendulum, this 'touch' differed in force, for long sub-sampling periods even a light touch could destabilize the process.

One obvious problem when starting the trails was the sensitivity to calibration error, therefore a lot of effort was put in to calibrate the sensors correctly before any trail yielding a just comparison base for all the controllers tried.

6.1 No sub-sampling - results

In this section only results with no sub-sampling will be discussed.
Having set the poles, the controllers were tested and the performance was recorded. Here are the results presented:

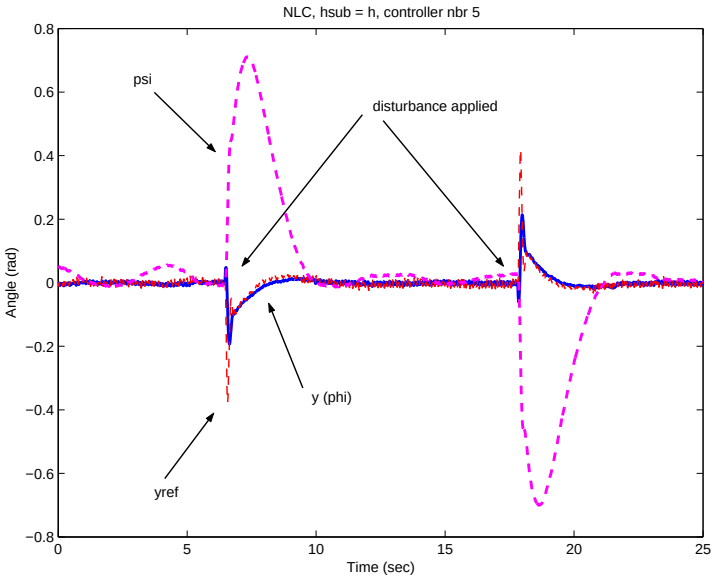

Figure 6.1: Plot of $y = \phi$, y_{ref} and ψ $\epsilon = 0.05$ using the NLC controller number 5.

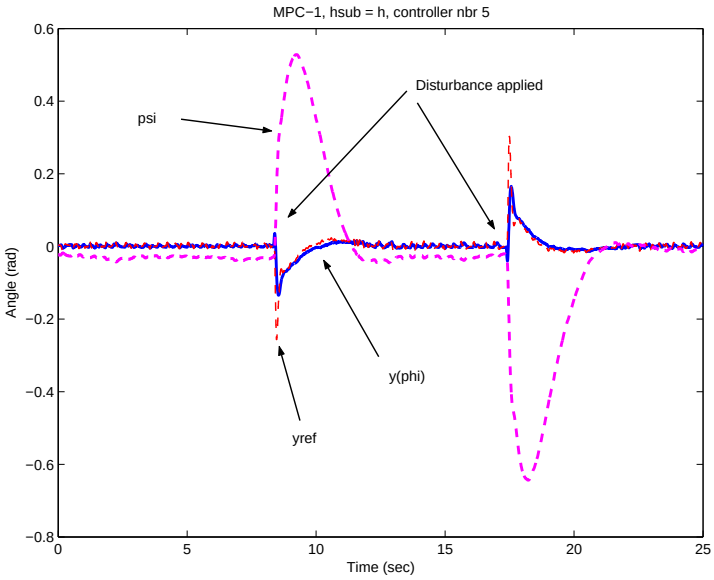

Figure 6.2: Plot of $y = \phi$, y_{ref} and ψ $\epsilon = 0.05$ using MPC controller number 5.

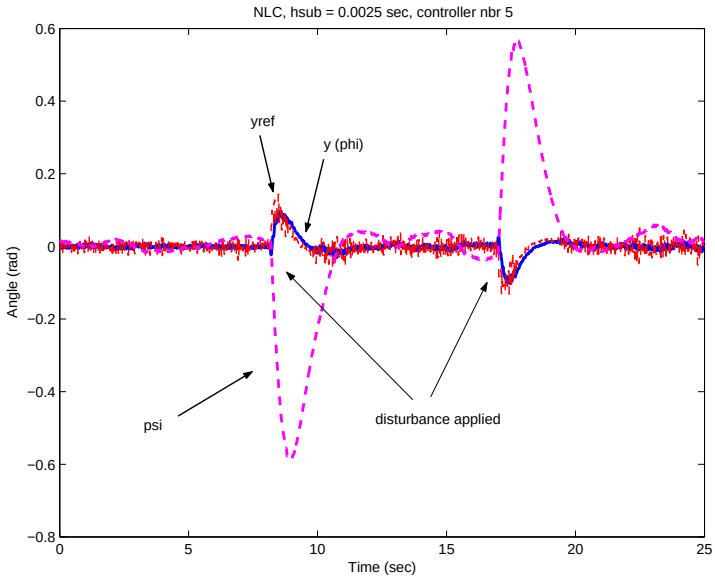

Figure 6.3: NLC under sub-sampling = 0.025 sec using weighting matrices as in controller 5

Notice the "dip" when tracking the reference, ϕ is going the other way before it follows the reference, this is a typical feature for non-minimum phase systems.
Here we can see that the controllers perform almost equally good with very good disturbance rejection and good recovery.

6.2 $\mathbf{h_{sub} = 0.025}$ - results

Using h_{sub} five times longer than the inner loop sampling period allows the optimization sequence to have longer time to finish and therefore hopefully a more exact solution. Using a longer sub-sampling period also allows us to evaluate MPC-2 which have been discussed previously.

For the controllers in this section $\epsilon = 0.05$ is used and for the MPCs a prediction horizon $T_{pred} = 0.5$ is used. Even though T_{pred} seems quite long compared to the sub-sampling period it is necessary due to the non-minimum phase features of the dynamics.

Once again the controllers seem to work similar. The extra time to solve the the optimization doesn't seem to do any significant difference.

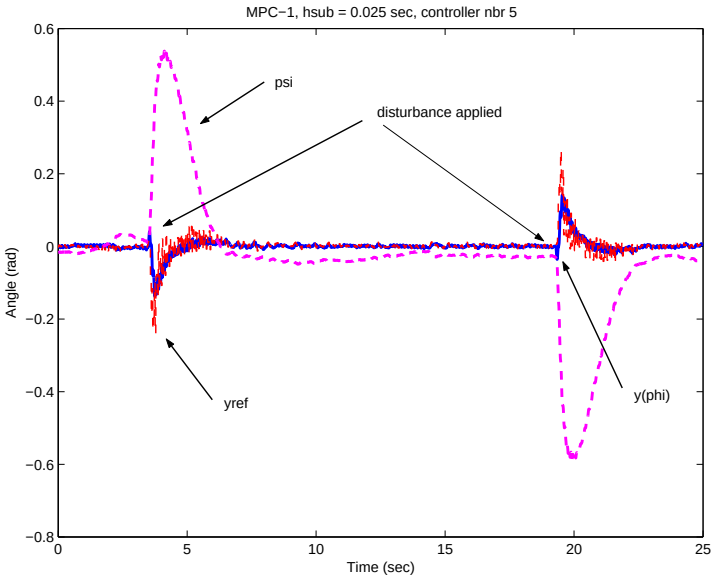

Figure 6.4: MPC-1 under sub-sampling = 0.025 sec using weighting matrices as in controller 5

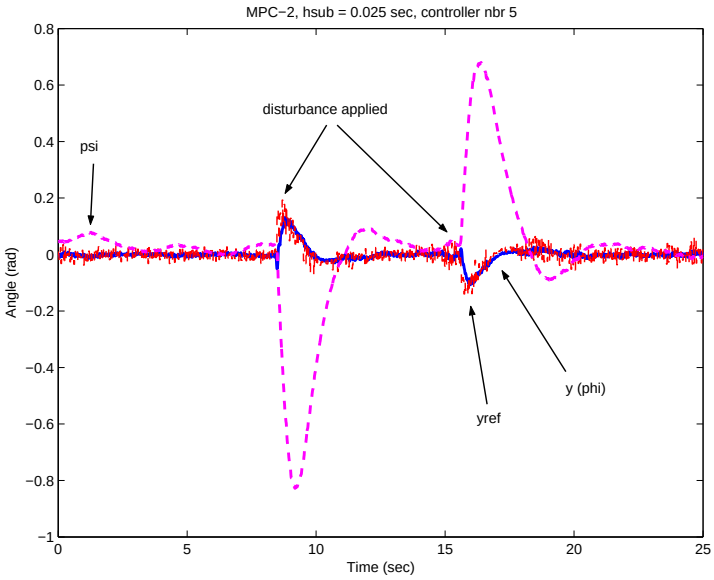

Figure 6.5: MPC-2 under sub-sampling = 0.025 sec using weighting matrices as in controller 5

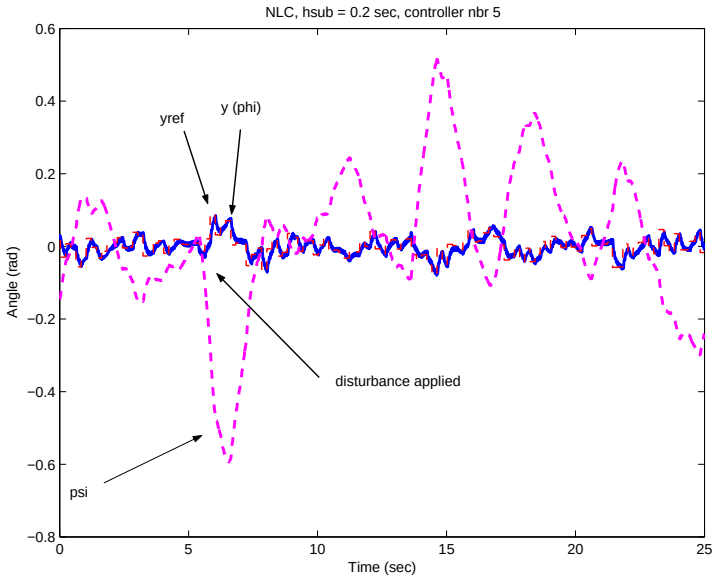

Figure 6.6: NLC under sub-sampling = 0.2 sec (controller 5)

The only reason to go further increasing the sub-sampling period is to investigate how the system will behave when the time-scale separation have been augmented i.e the outer loop will generate steps that will be reached by the inner loop in time.

6.3 $\mathbf{h_{sub}} = \mathbf{0.2}$ sec - results

Here the discretization is more evident, using the same weighting matrices as for $h_{sub} = 0.025$ yields different gains in this case. Controller 5 from table 5.6 was never used for the MPC-2 due to it was to slow, the arm reached the singularity area and was therefore not controllable. This made it hard to compare MPC-2 with the NLC and MPC-1.

The results show that NLC produces oscillations with higher frequency and amplitude than MPC-1. The delay between the feedback and the actual application of the control input is the reason for this behavior, the controller simply lags behind the system. Here the MPC-1 seems to be able to manage these delays fairly well.
The MPC-2 is hard to compare with any other controller due to it's totally different weighting matrices.

The disturbance rejection and recovery was bad for the NLC which was

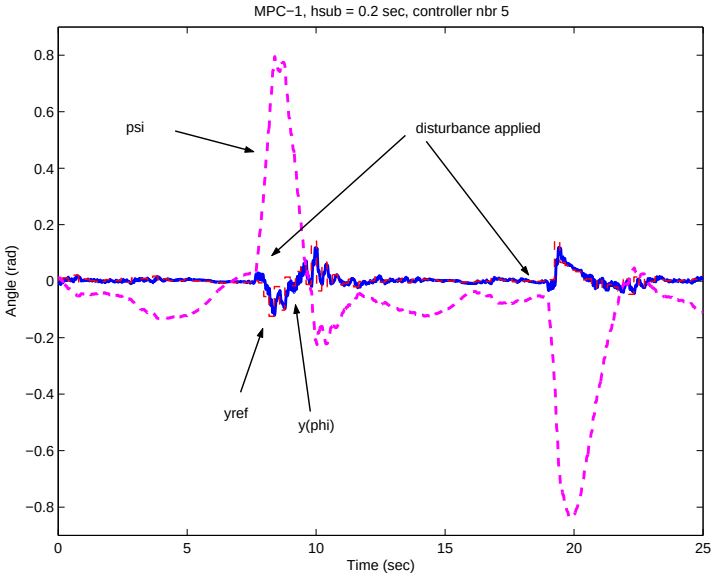

Figure 6.7: MPC-1 under sub-sampling = 0.2 sec (controller 5)

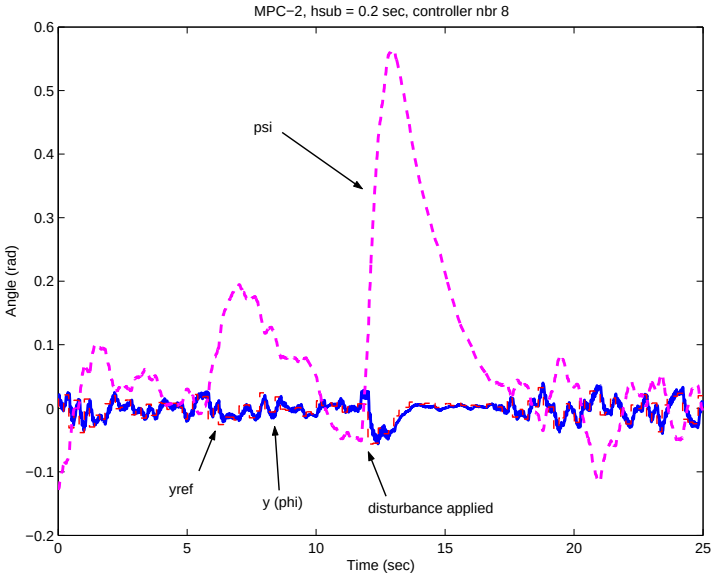

Figure 6.8: MPC-2 under sub-sampling = 0.2 sec (controller 8)

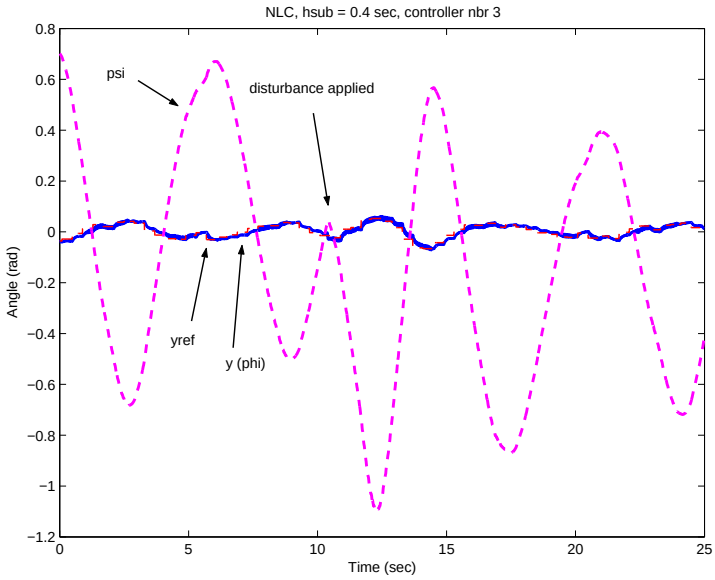

Figure 6.9: NLC -controller under sub-sampling = 0.4 sec (controller 3)

already suffering from heavy oscillations. However the MPCs could still deal with some disturbance quite good.

6.4 $\mathbf{h_{sub} = 0.4}$ sec - results

Using this sub-sampling frequency should allow the inner loop to reach the desired output within one sub-sampling period, hence the time-scaling should be enough.

As perhaps expected from the previous results using $h_{sub} = 0.2$ the NLC-controlled system oscillated heavily. The NLC showed some ability to reject or recover from disturbances even though this ability was quite small.

MPC-1 showed compared to the NLC better disturbance rejection and recovery abilities even though it's clear that long sub-sampling periods decreases this ability. MPC-2 was basically incomparable with the other controllers.

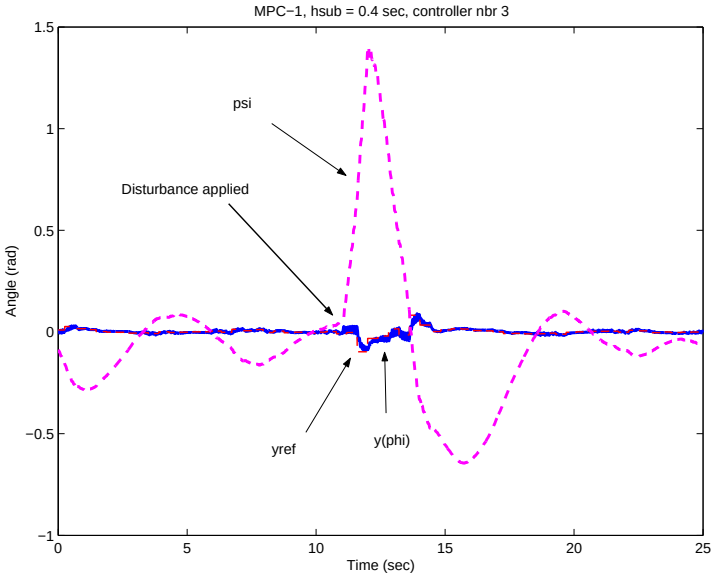

Figure 6.10: MPC-1 -controller under sub-sampling = 0.4 sec (controller 3)

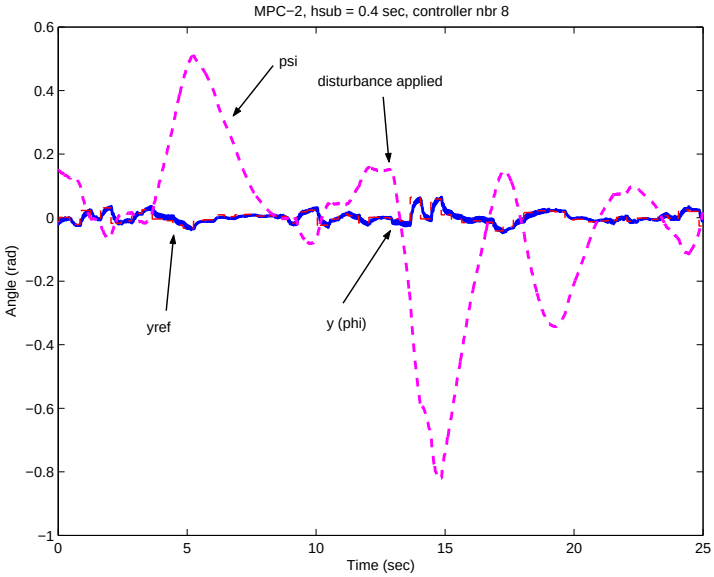

Figure 6.11: MPC-2 under sub-sampling = 0.4 (controller 8)

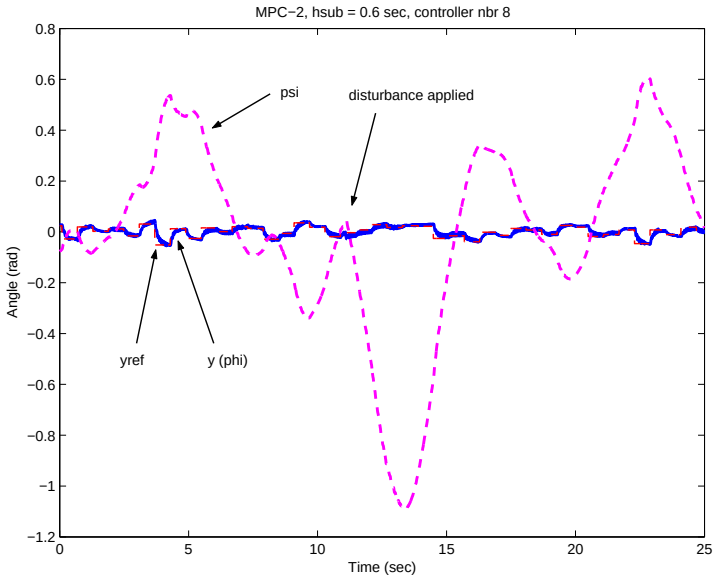

Figure 6.12: MPC-2 under sub-sampling = 0.6 sec (controller 8)

6.5 $\mathbf{h_{sub} = 0.6}$ - results

Having increased the sub-sampling period even further a stable NLC was simply not found. A sometime stable MPC-1 was found but the oscillations were of that magnitude that it can basically be considered as unstable. Here the MPC-1 and MPC-2 showed different features, while MPC-1 only worked when choosing lower than before weighting matrix, MPC-2 seems to work better when augmenting the weighting matrix. Because of the poor performance of the NLC and MPC-1 only the results from MPC-2 will be presented.

The disturbance rejection and recovery is very weak at this sub-sampling rate but is nevertheless still existent.
As observed in the figure, the system oscillates quite heavily but is still stable.
Even though these result are quite useless when only examine the performance, one interesting fact can be found: using MPC-2 which has a two dimensional λ seems still be stable for long sampling periods as 0.6 seconds.

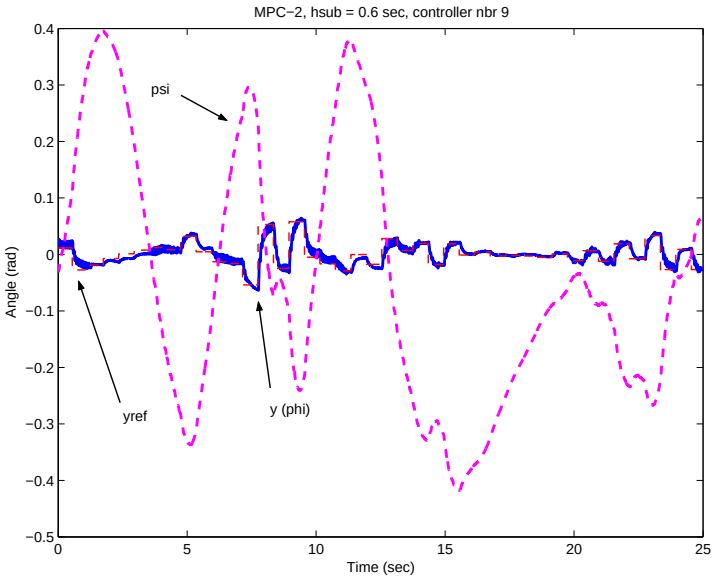

Figure 6.13: MPC-2 under sub-sampling = 0.6 sec (controller 9)

6.6 Estimating the feedback gains using least square method.

One alternative method to analyze the different results from the controllers used is to compute the estimate of the feedback gains using least square method. Assuming that the system is totally linear we can write the following equations:

$$y_{ref} = -K \cdot \begin{bmatrix} \bar{\eta}_1 \\ \eta_2 \end{bmatrix} \tag{6.1}$$

this illustrates an ordinary feedback control law $u = -Kx$. Having the measured values of $y_{ref} = [y_{ref}(1)\ y_{ref}(2) \ldots y_{ref}(N-1)\ y_{ref}(N)]$ and $\bar{\eta}_1 = [\bar{\eta}_1(1)\ \bar{\eta}_1(2) \ldots \bar{\eta}_1(N-1)\ \bar{\eta}_1(N)]$ and $\eta_2 = [\eta_2(1)\ \eta_2(2) \ldots \eta_2(N-1)\ \eta_2(N)]$, N is in our case 5000 measuring points. Using this we can estimate the feedback gain $\hat{K}$.

$$\hat{K} = -y_{ref} \cdot \begin{bmatrix} \bar{\eta}_1 \\ \eta_2 \end{bmatrix}^T \cdot \left(\begin{bmatrix} \bar{\eta}_1 \\ \eta_2 \end{bmatrix} \cdot \begin{bmatrix} \bar{\eta}_1 \\ \eta_2 \end{bmatrix}^T \right)^{-1} \tag{6.2}$$

How can these estimates be interpreted?, well these are the feedback gains when applied on a system without sub-sampling the system will show the

same features as if it was sub-sampled, not exactly however.

Estimate feedback gains when not sub-sampling, Table 6.1

$h_{sub}(sec)$	Cont.nbr	K_{NLC}	K_{MPC-1}	K_{MPC-2}
0.005	5	[3.18 5.33]	[3.18 5.11]	
0.025	5	[3.65 3.30]	[3.41 2.43]	[3.95 3.55]
0.2	5	[3.70 2.52]	[3.29 2.29]	
	8			[1.94 2.73]
0.4	3	[2.49 1.10]	[1.87 1.49]	
	8			[2.02 1.77]
0.6	8			[1.58 1.49]
	9			[2.29 2.02]

Table 6.1: Estimate feedback gains with no sub-sampling

Judging from the table above, we can actually compare the MPCs with the NLCs when it comes to comparing the 'average' feedback gains. This suggest that using Quasi-Newton in a way as it is done in MPC-2 is not completely valid due to the missing endpoint penalization, it's not really missing but rather never affected by the input we are using.

This discovery also lead us back to the use of the feedforward gain used in the MPC-2's, this feedforward gains is not needed in MPC-1. The feedforward gain can therefore also be used as a mean of compensation for not penalizing the endpoint in MPC-2.

6.7 Discussing the overall results

It might seem strange to include the results from the long sub-sampling periods when the using short sub-sampling periods clearly was superior. The reason doing this was to somehow grasp the features of the MPC that couldn't be noticed using shorter sub-sampling periods. According to previous simulations the use of a sub-sampling period of 0.4 seconds should have yielded a sufficient time-scale separation for the MPC to work correctly. The NLC proved perfect functioning despite the lack of ability to reach the setpoint after each sup-sampling period.

Comparing the NLCs and MPC-1s is quite easy, both show the same

features just as expected. Comparing the results with that of MPC-2 can however be more difficult because in some cases seemingly totally different controllers have to be compared with each other, despite of this, some general conclusions can be made.

Firstly even though it isn't totally clear in the plots but the NLC works smoother for short sub-sampling periods than the MPC-1s. this difference is so small that it is almost negligible. But for longer sub-sampling periods the delay is acting destabilizing yielding oscillations for the NLC while the MPC-1 seemingly still performed quite well even for longer sub-sampling periods. As seen in the results making a comparison between the NLC and the MPC-2 was almost impossible for longer sampling periods, although it can be said that MPC-2 showed interesting stability features after evaluating the results when using longer sub-sampling periods (0.4 and 0.6).

Another rather obvious difference between the NLCs and the MPCs is the time required to calculate an input signal, here when the NLC is used, it takes approximately $150\mu s$ to get a control signal, while compared the MPC-2 which have to optimize, it took up to $30000\mu s$ to compute an input signal. MPC-1 showed impressive speeds when optimizing, it took approximately $500\mu s$ to get a control signal, due to the rather simple optimization hence a solution was found rather quickly.

This difference in computing time pretty much defines where we can use the NLCs and the MPCs.
Hence we can generally say that if the dynamics seem fast or the sub-sampling frequency is fast it might be better to use a NLC or State-feedback structured controllers with fixed gains, since it doesn't require a online optimization which takes time. On the other hand if the dynamics are quite slow and the delay is long or the sub-sampling frequency is low for some reason, which mean longer computation time is available, maybe one of the MPCs will perform better. Also even though it haven't been implemented in this project, a MPC, according to theory deals better with constraints than the NLC since it's possible to include these in the optimization. Non-linear functions can also be optimized using online optimization just as shown in appendix A.2.

One difference that can be considered rather peculiar is that MPC-1 and MPC-2 differs so much in practice. This is due to the way they have been implemented, MPC-2 optimizes using 2 λ:s but only the first will be applied on the system. The applied λ is not affected by the final point penalization matrix, hence only the Q-matrix affect λ. MPC-1 on the other hand is affected by the final point penalization matrix and therefore it yields

another control signal, making it very hard to compare even the two MPCs.

The only, rather blunt, comparison that can be made is that MPC-1 shows average performance for the whole range, it work for when using no sub-sampling and it also works for when using rather long sub-sampling. An impressive result is that the MPC-1 performs equally good as the NLC when not sub-sampling.
MPC-2 is not able to work good with no sub-sampling but shows the best performance for long sub-sampling periods.

Even though the outer-loop controllers weren't truly global, no constraints was implemented in the optimization making the controller avoiding singularity areas, the NLC still showed that it can control better than a linear controller especially when the angular velocities $\dot{\psi}$ and $\dot{\phi}$ were not equal to zero. This became clear when using a swing up controller on the pendulum, the NLC could 'catch' the pendulum at a higher angular velocity of the pendulum ($\dot{\phi}$) than any linear controller. The 'best' LQR managed to catch the pendulum at $\dot{\phi} = 4.9$ (rad/s), the NLC could catch the pendulum at $\dot{\phi} = 6.2$ (rad/s).

Another advantage with the cascade scheme is the use of non-linear techniques for control design. Using non-linear techniques allows us to implement a global controller. The advantage of having a global controller arises when compared with an ordinary linear controller for example a LQR-controller, The linear controller is not performing well when far from the linearized point where it is designed around.

6.7.1 Future work suggestions

Suggestions for future work is to implement the cascade scheme on a inverted pendulum without singularity points, yielding that no or quite easy constraints in the optimization have to be considered, making it easier to implement a global controller.
Another suggestion is to improve the already existing MPC to include constraints.
Also developing the MPC to compute a sequence of control inputs (y_{ref}) instead of as now that the input (y_{ref}) is piecewise constant during the sub-sampling period.
Future work dealing with robustness analysis of the Pendubot using NLC and MPC is welcome as well.

Appendix A

A.1 the LQR-controller

Solving problem 3.9 analytically with an infinitive prediction horizon i.e $T = \infty$, yields an ordinary LQR-problem with a constant state-feedback gain matrix $\mathbf{K_{LQR}}$, where:

$$K_{LQR} = \begin{bmatrix} K_{LQR_1} & K_{LQR_2} \end{bmatrix} \tag{A.1}$$

Giving the control law:

$$\lambda = -\mathbf{K_{LQR}} \cdot \eta \tag{A.2}$$

Solving problem 3.9 analytically gives us:
Introducing extra state variables η_0 and η_3, since this is a non-autonomous problem, satisfying the state equation

$$\dot{\eta}_\mathbf{0} = \eta^\mathbf{T}\mathbf{P}\dot{\eta} + \frac{\mathbf{1}}{\mathbf{2}}\eta^\mathbf{T}\mathbf{Q}\eta + \frac{\mathbf{1}}{\mathbf{2}}\mathbf{R}\lambda^\mathbf{2}, \quad \eta_0(t_0) = \frac{\mathbf{1}}{\mathbf{2}}\eta^{\mathbf{0T}}\mathbf{P}\eta^\mathbf{0} \tag{A.3}$$

and

$$\dot{\eta}_3 = 1, \quad \eta_3(t_0) = t_0 \tag{A.4}$$

The Hamiltonian:

$$H = \frac{1}{2}z_0(\eta^\mathbf{T}\mathbf{Q}\eta + \mathbf{R}\lambda^\mathbf{2}) + \mathbf{z}^\mathbf{T}(\mathbf{A}\eta + \mathbf{B}\lambda) + z_3 \tag{A.5}$$

The co-state equations are:

$$\dot{z}_0 = 0, \quad \dot{\mathbf{z}} = -z_0\mathbf{Q}\eta - \mathbf{A}^\mathbf{T}\mathbf{z}, \quad \dot{z}_3 = -\frac{\partial H}{\partial \eta_3} = -\frac{\partial H}{\partial t} \tag{A.6}$$

The find the minimum we set $\frac{\partial H}{\partial \lambda} = 0$, using $z_0 = -1$. Giving us the supremum $-\mathbf{R}\lambda + \mathbf{B^T z} = \mathbf{0}$ giving us the control law $\lambda = \mathbf{R^{-1}B^T z}$. The dynamics of the states and co-states are now:

$$\dot{\eta} = \mathbf{A}\eta + \mathbf{B}\mathbf{R^{-1}}\mathbf{B^T z},\ \dot{\mathbf{z}} = \mathbf{Q}\eta - \mathbf{A^T z} \tag{A.7}$$

For a fixed time problem, as in our case the conditions for H vanishes along the optimal trajectory, but since the target is free the transversality condition $\mathbf{z^1} + \mathbf{P}\eta^\mathbf{1}$ have to be satisfied letting us assume that $\mathbf{z} = -\mathbf{K}(\mathbf{t}, \mathbf{t_1})\eta$, the transversality condition then becomes $\mathbf{K}(\mathbf{t_1}, \mathbf{t_1}) = \mathbf{P}$ differentiating $\mathbf{z} = -\mathbf{K}(\mathbf{t}, \mathbf{t_1})\eta$ and manipulating the result gives us:

$$\dot{\mathbf{K}} = \mathbf{K}\mathbf{B}\mathbf{R^{-1}}\mathbf{B^T K} - \mathbf{K}\mathbf{A} - \mathbf{A^T K} - \mathbf{Q} \tag{A.8}$$

and since the prediction horizon $T = \infty$ we can assume that K will converge this gives us $\dot{K} = 0$. Solving equation A.8 with $\dot{K} = 0$ gives us the control law $\lambda = -\mathbf{R^{-1}B^T K}\eta$ where the feedback gains are $\mathbf{K_{LQR}} = \mathbf{R^{-1}B^T K}$, where $y_{ref} = \arcsin(\lambda)$. This is all done quickly in `MATLAB` using the command `LQR.m`. The LQR feedback gains can be used for the PD-controller as well as for the NLS.

A.2 The non-linear MPC

A MPC using non-linear reduced internal dynamics instead of linear ones was called the non-linear MPC. The non-linear dynamics caused a more time demanding optimization than linear ones and the effort was therefore concentrated on the linear set of reduced internal dynamics instead. The non-linear MPC used a 2-dimensional λ and RK4 as integration method. RK4 was needed to improve the prediction precision.
The reduced internal states used by the non-linear MPC:

$$\begin{aligned} \bar{\eta}_1 &= \psi \\ \eta_2 &= J\dot{\psi}\cos(\psi - \phi) + J_2\dot{\phi} \end{aligned} \tag{A.9}$$

The reduced internal dynamics used by the non-linear MPC:

$$\ddot{\phi} = v$$

$$\begin{aligned} \dot{\bar{\eta}}_1 &= \frac{\eta_2}{J\cos(\eta_1 - y_{ref})} \\ \dot{\eta}_2 &= g_2 \sin(y_{ref}) \end{aligned} \tag{A.10}$$

The results of the non-linear MPC are presented below.
The interesting point about using the non-linear dynamics was to see if the Quasi-Newton optimization algorithm could deal with non-linear dynamics. One constraint was significant for when this MPC could actually be used, this constraint was the number of optimization iterations per sampling period. The non-linear MPC could only do 2 iterations per sampling period which is quite slow compared to 20 for the other MPCs using linear RIS's, hence `nlMPC`, which is the name of the C-function for non-linear MPC, could only be used when h_{sub} was sufficiently large, the non-linear MPC had to be able to do around 200 iterations per sub-sampling period (160 when $h_{sub} = 0.4sec$ and 240 when $h_{sub} = 0.6sec$).

By linearizing the internal states for the linear reduced internal dynamics as in **??** we can see that there is a factor difference of J (in table 2.1) between the reduced internal states η_1 and η_1 yielding that the weighting matrices in table 5.6 still can be used just by changing the linearized system matrices and the weighting matrices accordingly: Linearizing dynamics in 3.4 gives:

$$\begin{aligned} \dot{\eta}_1 &= \frac{\eta_2}{J} \\ \dot{\eta}_2 &= g_2 \lambda \end{aligned} \tag{A.11}$$

knowing that $\eta_1 \approx J\bar{\eta}_1$ when linearizing $\bar{\eta}_1$, yields the weighting matrices for the non-linear MPC:

$$Q_{nonlin} = \begin{bmatrix} J^2 & 0 \\ 0 & 1 \end{bmatrix} \cdot Q_{lin} \tag{A.12}$$

Using dynamics A.11, equation A.12 and controller 8 from table 5.6 a controller was designed and tested for two different sub-sampling periods: 0.4 and 0.6 seconds, the prediction horizon was 0.8 respective 1.2 seconds. Figures A.1 and A.2 illustrates the plotted results. As figure A.1 and A.2 suggests, that the non-linear MPC (`nlMPC`) have equivalent performance as the linear MPC-2 (`MPC-2`), just as expected.

A.3 Swing up

Even though the swing up wasn't necessary in this project it was implemented for demonstration purposes. Also the swing up could be used when

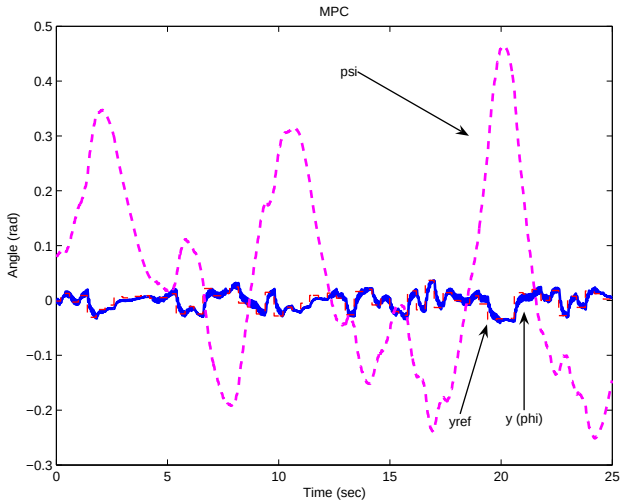

Figure A.1: non-linear MPC under sub-sampling = 0.4 sec

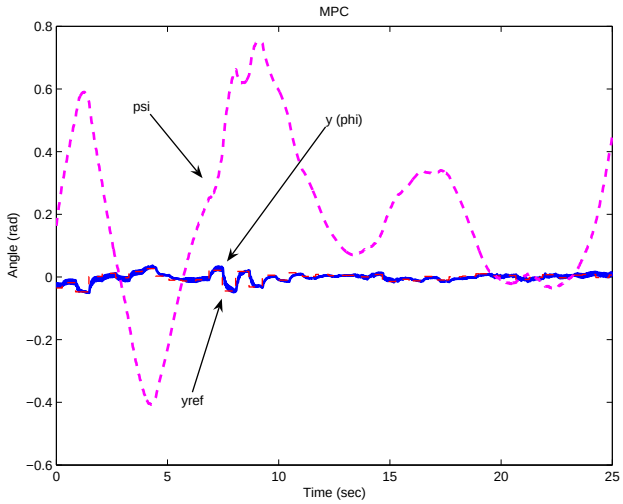

Figure A.2: non-linear MPC under sub-sampling = 0.6 sec

investigating were the limitation were for catching the pendulum.

Two swing ups were implemented, a simple swing up (simple swing up) and a more advanced one using IOFL, named `SwingUpController`.

The simple-swing up worked accordingly: Swing the arm back and forth until enough energy is in the system for the pendulum to turn around full circles. When the pendulum is turning around full circles the swing up waits until the pendulum have a speed at which it is possible to catch it.

The IOFL-swing up used IOFL techniques to yield the system linear using the ψ-angle as output instead of the ϕ-angle. A ϕ-angle destabilizing control input was calculated to pump in energy into the system. When the ϕ-angle was pointing upwards the controller made an attempt to catch it.

The swing ups were mainly used when no sub-sampling was used and when using the NLC. When sub-sampling the controllers had rather poor 'catching' abilities.

Appendix B

Bibliography

Bibliography

[1] K. Guemghar, B. Srinivasa, Ph Mullhaupt and D. Bonvin. *Control of nonlinear nonminimum phase systems using trajectory generation.* Nelcos 2003.

[2] K. Guemghar, B. Srinivasa, and D. Bonvin *A Cascade Structure with Predictive Control and Feedback Linearization for the Control of Nonlinear Systems.* IEEE03.

[3] J.Y. Favez. *Commande d'un pendule polaire.* Ecole Polytechnique Fdrale de Lausanne-Switzerland, 1998.

[4] L. M. Hocking. *Optimal Control An Introduction to the Theory with Applications.* Clarendon press, Oxford, 1991.

[5] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer - Verlag New York, Inc, 1999.

[6] W. H Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery. *NUMERICAL RECIPES in C.* Cambridge University Press, 1992.

[7] J. kesson. *Safe Manual Control of Unstable Systems.* Department of Automatic Control Lund Institute of Technology, September 2000.

[8] D. Henriksson and J. kesson. *Flexible Implementation of Model Predictive Control Using Sub-Optimal Solutions.* Department of Automatic Control Lund Institute of Technology, April 2000.

[9] A. Isodori. *Nonlinear Control Systems.* 2nd edition, Springer-Verlag Berlin, 1989.

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