New method for solving iterative learning control problems for non-minimum phase systems

New method for solving iterative learning control problems for non-minimum phase systems

A.P.Teerhuis

DCT 2004.80

Traineeship report

Supervisor: M.J.G. van de Molengraft

Technische Universiteit Eindhoven
Department Mechanical Engineering
Dynamics and Control Technology Group

Eindhoven, May 26, 2004

Abstract

In modern industry, production machines form the basis for manufacturing mass products. A lot of these machines have only one task, which they'll perform over and over again. Time and costs are essential, so the better and faster it works, the more money you make. To make such machines perform better, a good knowledge of its dynamics is needed. With this knowledge, good control strategies can be developed. Another way to make the machine perform better, is to let it learn from his earlier mistakes. This is called Learning Control. A lot of research has been done already, and a lot of solutions were found. Non-minimum phase systems, however, form a big problem when solving with Learning Control, because the positive zeros become unstable poles in the learning filter. In this report a new solving method is discussed and compared with a conventional method. This new method can deal with Learning Filters which contains more zeros than poles and also it solves for unstable poles. The general idea is to split the Learning Filter into a causal and non-causal part and filter each part separately.

Contents

Chapter 1

Introduction

Today's mass production processes mostly consist of machines doing the same job over and over again. Speed and accuracy are keywords for these kind of processes. A good knowledge of the system is required to implement new and better control strategies. One of those strategies is learning control.

1.1 What is (iterative) learning control?

Learning control is a type of control in which the controller learns from its own previous mistakes.

When a purely feedback controlled system (e.g. figure 1.1) performs the

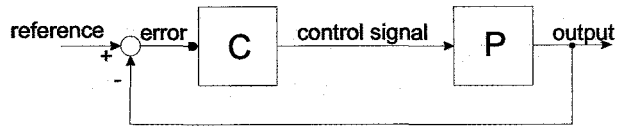

Figure 1.1: *Standard feedback system*

same (array of) actions over and over again, one can imagine that the signals which are used to control the actuators will (globally) have great similarities each sequence. As a matter of fact, a 'base' signal (see figure 1.2) can be extracted from several sequences. Any differences from this base signal are of stochastic nature or can be attributed to environmental issues.

The base signal can be passed to the system as a feedforward signal. This feedforward signal is responsible almost completely for the positioning. Any deviations from the reference signal will be much smaller than would have been the case for a purely feedback system, meaning that the controller can focus on other sources of errors. The block diagram shown in figure 1.1 needs to be adjusted to the the new feedforward configuration, as presented in figure 1.3.

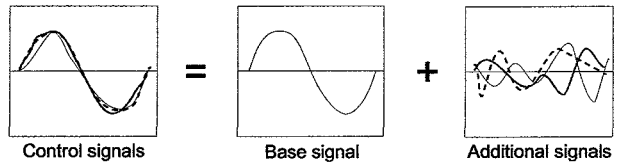

Figure 1.2: *Extracting a base signal from several control signals*

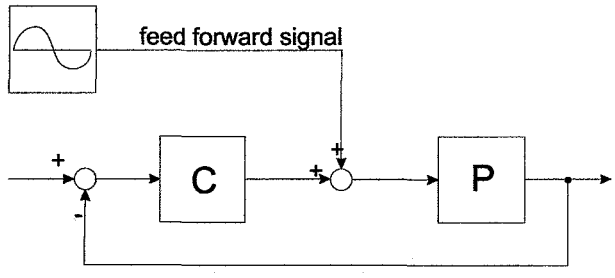

Figure 1.3: *Feedforward system with base signal*

The feedforward can also be calculated from (experimental) data, instead of 'extracted' from control signals. This can be done when both the system characteristics and the tracking error(s) are known. When future experiments or production cycles, using the feedforward, point out that there is still a definite correlation in the new tracking error(s), a new iteration step can be made. The new calculated feedforward signal is the signal that rules out the tracking errors for which it was calculated, therefore it has to be added to the already existing feedforward. If certain user specified demands are met or purely stochastic tracking errors remain, the iteration process can be stopped and the found feedforward signal is the best you can achieve for the system. The method described above is presented schematically in figure 1.4

In figure 1.4, S_p^{-1} is the inverse process sensitivity defined as:

$$\frac{e}{\Delta f} = \frac{P}{1+PC} = S_p \tag{1.1}$$

In equation 1.1, Δf is the signal that is to eliminate the last found tracking error. It can be written in following way:

$$\Delta f = S_p^{-1} e = L.e \tag{1.2}$$

Here, L is referred to as the learning filter. After this, the new (total) feedforward is the summation of the old feedforward with the Δf calculated from equation (1.2):

$$f_{new} = f_{old} + \Delta f \tag{1.3}$$

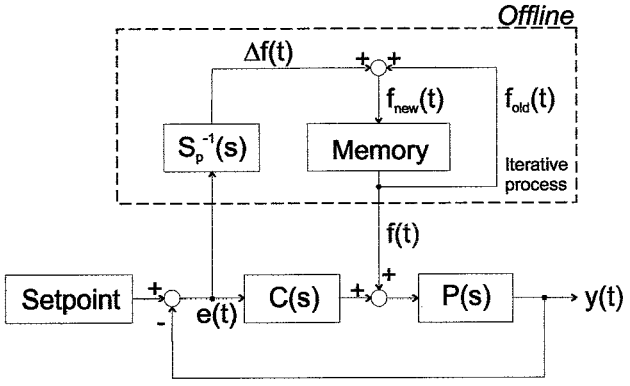

Figure 1.4: *Schematic view of the learning algorithm*

The algorithm discussed above is, as already mentioned, an iterative process, one can repeat until the demands have been reached or when there are no more dynamics left in the tracking error, due to measurement or stochastic noise.

Notice that when no control action is present (the process is steered), the tracking error $e(t)$ will be equal to the reference and the learning filter $L(s)$ will be the exact inverse of the system $P(s)$.

A major advantage of ILC is that without a perfect knowledge of the system, still very good results can be obtained after few iterations [1].

1.2 Goal of this research

The global concept of learning control is discussed in the previous section. When implementing it for real systems, one can encounter all different kinds of problems.

The inverse of the process sensitivity is needed to calculate feedforward signals. For non-minimum phase systems, this means that the inverse becomes (partially) unstable.

Several algorithms have been developed. One of the most used algorithms is ZPETC. Recently, another algorithm is proposed. Both algorithms will be discussed in further detail in chapter 2. In this report, a comparison between the two algorithms will be made.

For the case of production processes doing endlessly the same (array of) movements, one can imagine that the same feedforward signal is therefore repeated each time. For reasons of continuity the end-conditions of the feedforward have to be the same as the begin-conditions of the following (new) feedforward. The algorithm calculating the feedforward has to account for these (two-point) boundary values.

Goal of this research is to present a framework within MATLAB to solve iterative learning control problems for non-minimum phase systems, taking bound-

ary values at both ends of the feedforward signal into account.

Chapter 2

Learning control methods

This chapter discusses the ZPETC method for calculating feedforward signals, as well as the new method. An example shows the capabilities of the two methods.

2.1 ZPETC

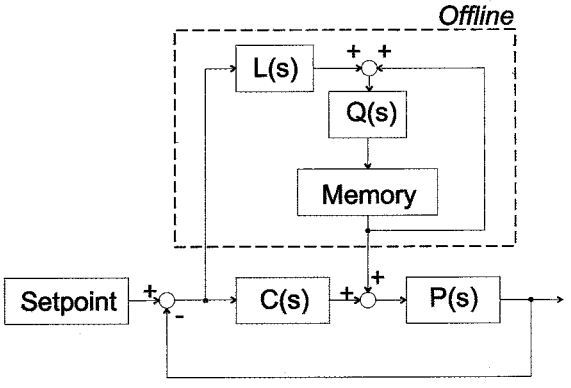

Figure 2.1: *Learning algorithm for use with ZPETC-method*

The key problem with learning control is to find a learning filter which is the inverse of the process sensitivity. Especially when dealing with non-minimum phase systems, the inverse can contain unstable poles. Since the error signal is filtered with the inverse, one wants to use a stable approximation of the inverse process sensitivity. This is done by a method called ZPETC (Zero Phase Error Tracking Control). The approximated learning filter is stable, so that there will be no problems when filtering the tracking error with it.

During the approximation process, the dynamics have been 'changed'. Due to these changes, an error signal filtered by the approximation will not give the same results as an exact inverse would do. For high frequencies, this becomes a problem, therefore a robustness filter $Q(s)$ is introduced. The Q-filter makes sure that the following convergence criterion is met:

$$\left| Q\left(1 - \frac{P}{1+PC}L\right)\right| < 1 \tag{2.1}$$

Here, L is the inverse approximation of the process sensitivity. Usually, the Q-filter has low-pass characteristics.

The filtered tracking error (i.e. the new feedforward update Δf) is filtered with this robustness filter. This has to be done anti-causally to compensate for the lead/lag of $Q(s)$. This method will be called the ZPETC-method throughout this report.

2.2 D&F

In some cases the process sensitivity can have more poles than zeros. As a result, the learning filter has more zeros than poles, making it more difficult to filter signals with this learning filter using standard routines. The ZPETC-method modifies the original inverse of the process sensitivity. In that case a perfect feedforward signal can not be found.

In [2] a new method is proposed, especially designed for systems having a numerator of higher order than the denominator. This is presented in the following equation:

$$L = S_P^{-1} = \frac{\prod_{i=1}^{p}(s - z_i)}{\prod_{j=1}^{q}(s - p_j)}, \quad p > q \tag{2.2}$$

The excess of zeros will be split from $L(s)$, resulting in a causal and a non-causal part, as can be seen in the next equation:

$$L = \underbrace{\prod_{k=1}^{p-q}(s - z_k)}_{\text{non-causal}} \underbrace{\frac{\prod_{i=1+(p-q)}^{p}(s - z_i)}{\prod_{j=1}^{q}(s - p_j)}}_{\text{causal}} = L_{nc}(s)L_c(s) \tag{2.3}$$

The non-causal part of equation 2.3 is a polynomial in s and in fact it's nothing less than a differentiating action.

With the split learning filter, the calculation now has two phases. The feedforward update Δf will be obtained by first filtering the tracking error with the

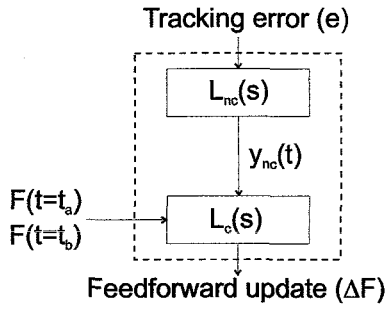

Figure 2.2: *Schematic representation of the new method*

non-causal part, resulting in a signal called $y_{nc}(t)$, and secondly, filtering with the causal part resulting in Δf. Visually, the method is presented in figure 2.2.

2.3 Learning control on a 4^{th} order system

In previous sections, the idea of (iterative) learning control is discussed. So far, no real example has been given. In this section we'll show the effectiveness of learning control by calculating the feedforward for a stable 4^{th} order system. Physically the example can be thought of as two masses connected to each other with a spring and a damper (see figure 2.3).

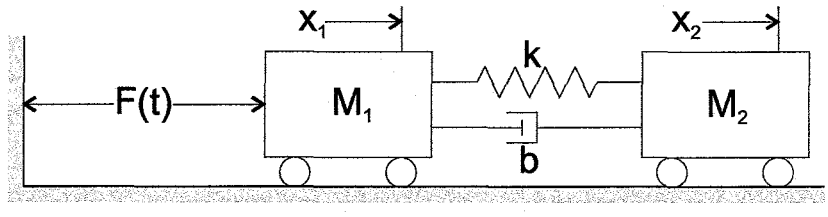

Figure 2.3: *Schematic view of the 4^{th} order system*

While measuring the position of the second mass, the first will be actuated. The transfer function is:

$$\frac{X_2(s)}{F(s)} = \frac{bs + k}{m_1 m_2 s^4 + b(m_1 + m_2)s^3 + k(m_1 + m_2)s^2} \tag{2.4}$$

In appendix A.1, the derivation of equation 2.4 is given, as well as all parameters used in this example. In this example a simple PD-controller will be used. It is not an optimal controller, because the main point of this report is learning control. Figure 2.4 shows the open loop nyquist diagram for this system.

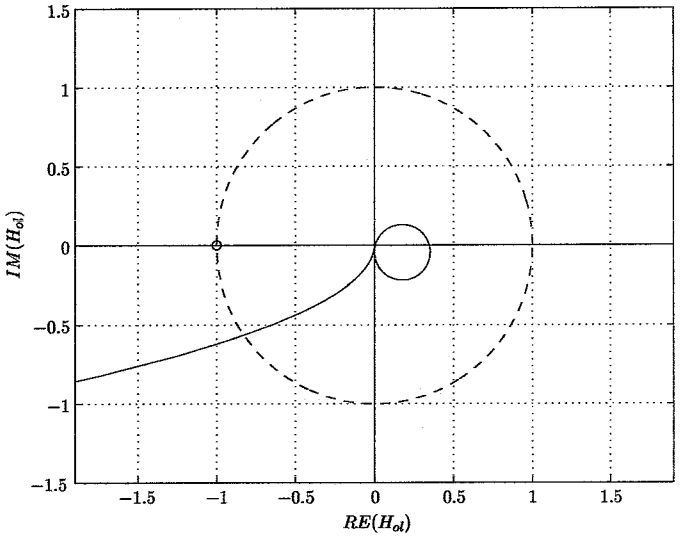

Figure 2.4: *Nyquist diagram of the open loop dynamics.*

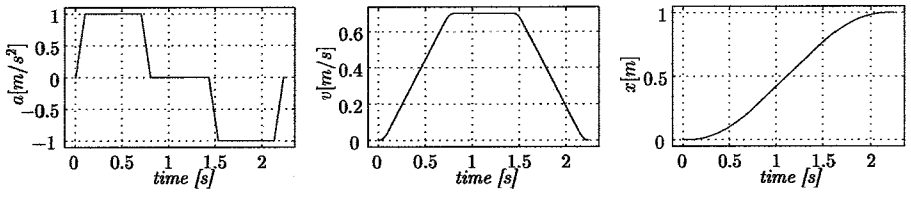

Figure 2.5: *Reference signal of the 4^{th} order system.*

The setpoint which will be presented to the system is a third order point-to-point movement as can be seen in figure 2.5.

No feedforward is given for the first simulation. The result of the tracking error can be seen in figure 2.6.

After just one iteration, the tracking error is reduced significantly for both the ZPETC-method, as well as the new D&F method, see figure 2.7.

From now on we will take the Integrated Absolute Error (IAE) as a measure for the performance of the system. The IAE is defined as:

$$IAE = \int_{t=t_0}^{t=t_1} e(t)dt \tag{2.5}$$

Figure 2.8 shows the IAE during 20 iterations. As can be seen from this figure, both methods score almost the same, except for the ZPETC without

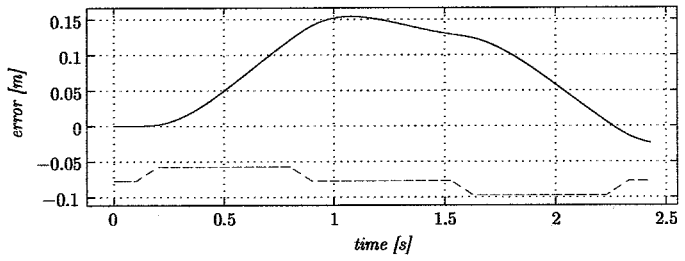

Figure 2.6: *Tracking error for the 4^{th} order system without using feedforward. Dashed line represents the acceleration setpoint.*

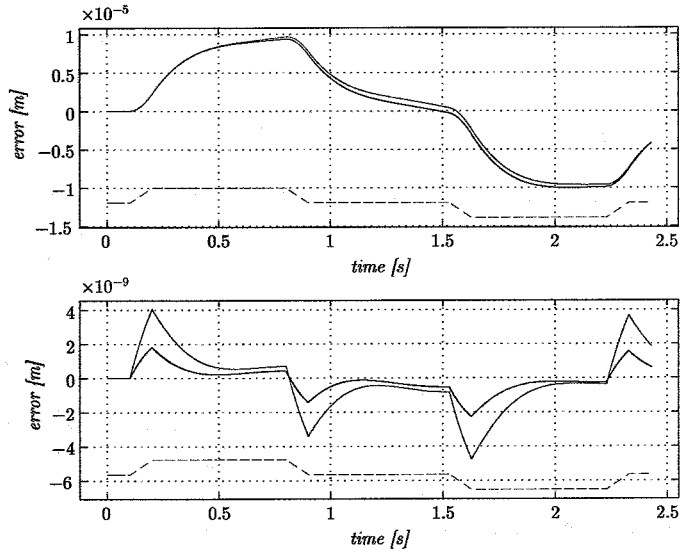

Figure 2.7: *Servo error for the 4^{th} order system using the ZPETC method (blue) and the D&F method (red) after the first (top) and second (bottom) iteration.*

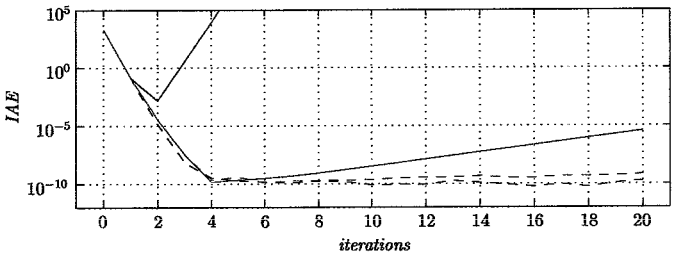

Figure 2.8: *Integrated Absolute Error for the 4^{th} order system using the ZPETC method (blue) and the D&F method (red). Without Q-filter (solid) and with Q-filter (dashed).*

a Q-filter. After a minimum IAE value has been reached, the D&F method without using the Q-filter starts to increase the IAE. However, when the same Q-filter as with the ZPETC method is used, both methods will perform the same.

The best feedforward and resulting tracking error, after the fourth iteration, for each method are presented in figure 2.9 .

As can be seen in figure 2.8, the servo error with the D&F method will increase after the fourth iteration step. This is due to numerical problems used for this learning problem. When decreasing the sample time used in this example, the right part of the line will flatten, to become (eventually) a horizontal line. The calculation time on the other hand will increase dramatically. For this example, where the inverse of the process sensitivity is stable, both methods almost perform the same. Later, when a non-minimum phase system will be discussed, we will see that the ZPETC method performs worse.

As stated before, the D&F method does not need a robustness filter, since the inverse of the process sensitivity is in essence exact. However, due to discretization, in order to calculate the feedforward signal, the dynamics will be altered slightly. The feedforward signal without using the Q-filter has a spiky character which leads to numerical problems for the used solver. Decreasing the step size and tolerances to zero will solve the problem, however, the calculation time will increase. Another way is to smooth the signal which is presented to the filter. Therefore the same Q-filter as for the ZPETC method is used.

2.4 Disadvantages and improvements

Learning control will be most likely used in (high-speed) mass production. Every separate movement can be optimized with this algorithm. Later, these optimized movements can be combined or 'glued' together forming an array of movements, however the conditions at the endpoint of one movement must be the same as the starting point of the next movement. Hence, when calculating the feedforward, one wants to specify both begin and end conditions (two-point boundary value

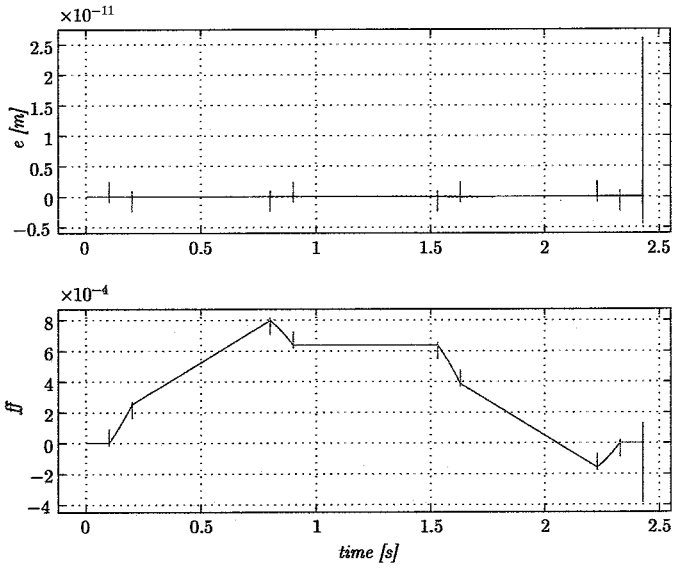

Figure 2.9: *Servo error (top) and feedforward signal (bottomt) for the 4^{th} order system using the ZPETC method (blue) and the D&F method (red).*

problem).

The previous example was a stable 4^{th} order system, also its inverse was stable, therefore the calculations on this system were not complicated. The problems start when the inverse dynamics of the system, which are directly in the learning filter, become unstable. This is the case with a non-minimum phase system.

Non-minimum phase systems have an inverse response, due to one or more zeros in the right half plane. When taking it's inverse, the positive zero becomes a positive, unstable pole, making the calculations with this learning filter more complicated. Earlier work on solving mixed stable/unstable systems resulted in a new ODE-solver [3]. This so-called RICLBVP (RICatti Linear Boundary Value Problem) is a two-point boundary value solver for stiff systems.

The next chapter discusses the design of a MatLab based program to solve non-minimum phase learning problems with a two-point boundary value problem using the RICLBVP-solver.

Chapter 3

Design Learning Control Tool in MATLAB

In this chapter, a framework is presented to solve learning control problems for different kind of systems. In this report we only focus on non-minimum phase systems with two-point boundary value demands, but in future, also other types of systems and algorithms can be integrated in this tool.

3.1 Splitting the system

As has been mentioned earlier, the learning filter can contain more zeros than poles, therefore the system has to be split into two parts (non-causal and causal part).

One has to be careful when splitting the system. Sometimes the poles of a system are complex conjugate pairs. These pairs must stay together to avoid calculation errors in MatLab. When splitting the system, the order of the non-causal (differentiating) part is always tried to be kept as low as possible, whereas the causal part is preferred to be strictly proper (ie. same order of numerator and denominator). In the case of complex conjugate pairs it can occur that the order of the numerator in the causal part is one less than the denominator. As a result of this, the order of the differential part the order increases by one. Appendix (B.1) shows the MatLab script and some comments on this method

3.2 Using differentiating filters

If the learning filter contains more zeros than poles, splitting the system will end up with a non-causal part which is not equal to 1. This part tells something about the way the input signal has to be differentiated. In [2] some differentiating mehtods (eg. Shannon-based differentiators and Polynomial interpolation/fitting) are discussed and implemented in MatLab.

These filter schemes results in imperfections at the begin and the end of the differentiated signal, due to the lack of data at these points. These border distortions are the major source of concern when using it for feedforward calculations.

Figure 3.1 shows the way in which the non-causal is implemented.

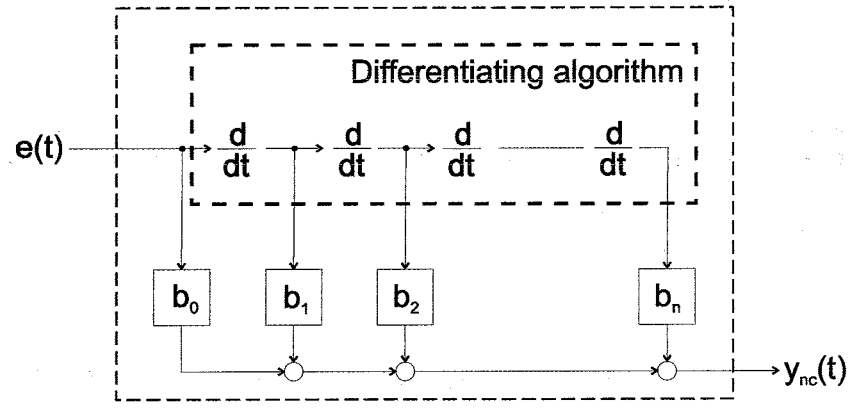

Figure 3.1: *Schematic representation of non-causal filtering*

3.3 Using a two-point value boundary solver for unstable systems

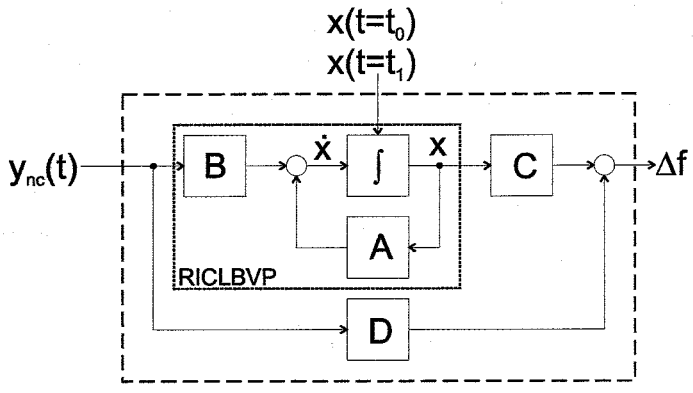

Figure 3.2: *Schematic representation of causal filtering*

The previous chapter already discussed that the inverse of (stable) non-minimum phase systems become unstable and are therefore rather difficult to

solve. One cannot use a standard MATLAB ODE-solver in these cases. In [3] a solver has been developed which deals with combined stable/unstable systems. Also a number of begin and end conditions can be accounted for.

The RICLBVP-solver does not give a direct solution for the feedforward update, it only solves the state at each time. From equation 3.1 this can be made clear:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{3.1}$$

Above, the general state-space notation is given. This represents the causal part of the learning filter. The signal $u(t)$ will, in this case, be the (non-causal) filtered error signal. The output $y(t)$ is the same as Δf, as mentioned in equation 1.3. Equation 3.1 now tranforms into the following equation (see figure 3.2):

$$\begin{aligned} \dot{x}(t) &= Ax(t) + By_{nc}(t) \\ \Delta f(t) &= Cx(t) + Dy_{nc}(t) \end{aligned} \tag{3.2}$$

The RICLBVP-solver calculates the state $x(t)$. In order to use the RICLBVP-solver to calculate ILC problems, the initial and the end conditions for the feedforward $\Delta f(t)$, must be rewritten for the state $x(t)$:

$$B_a \underline{x}(t_0) + B_b \underline{x}(t_1) = \underline{d}_x \tag{3.3}$$

In this equation, B_a and B_b are square matrices, used by the RICLBVP solver. The vectors $\underline{x}(t_0)$ and $\underline{x}(t_1)$ are the state vector boundary values at the begin and end point respectively and $\underline{d}(x)$ is the relation between these boundary values specified by the matrices B_a and B_b. This vector is also passed to the solver. Combined boundary values (eg. $\dot{x}(t_0) + \dot{x}(t_1) = 0$) cannot be dealt with by the RICLBVP-solver.

With learning control we do not want to specify begin and end conditions for the state $x(t)$ in figure 3.2, instead we want to specify the begin and end conditions on the feedforward signal $f(t)$. The learning algorithm, however, only calculates the feedforward update $\Delta f(t)$. The two-point boundary value problem for the feedforward update can be written in the same way as equation 3.3:

$$W_a \, \Delta \underline{f}(t_0) + W_b \, \Delta \underline{f}(t_1) = \underline{d}_{\Delta f} \tag{3.4}$$

When $\Delta f(t)$ is rewritten in terms of the state-vector $x(t)$ and the vector $y_{nc}(t)$ (see Appendix C) we get:

$$\begin{aligned} \Delta \underline{f}(t_0) &= O\underline{x}(t_0) + R\underline{y}_{nc}(t_0) \\ \Delta \underline{f}(t_1) &= O\underline{x}(t_1) + R\underline{y}_{nc}(t_1) \end{aligned} \tag{3.5}$$

Substituting into equation 3.4 this results in:

$$W_a O \underline{x}_{t_0} + W_b O \underline{x}_{t_1} = \underline{d}_{\Delta f} - W_a R \, \underline{y}_{nc,t_0} + W_b R \, \underline{y}_{nc,t_1} \tag{3.6}$$

From this equation, the matrices B_a and B_b as well as the vector $\underline{d}_x$ can be determined:

$$\begin{aligned} B_a &= W_a O \\ B_b &= W_b O \\ \underline{d}_x &= \underline{d}_{\Delta f} - W_a R\, \underline{y}_{nc,t_0} + W_b R\, \underline{y}_{nc,t_1} \end{aligned} \tag{3.7}$$

Now, the complete MATLAB tool for solving non-minimum phase feedforward signals is made, so we can use it to solve the iterative learning problem for which it is designed. The next chapter discusses the iterative learning problem for a non-minimum phase system with a two-point boundary problem.

Chapter 4

Comparison D&F-method vs. ZPETC

In this chapter, the new method will be compared with the ZPETC method for non-minimum phase systems. The goal of this example is to position the tip of a flexible beam.

4.1 Model of the flexible beam

In figure 4.1 a sketch of the flexible beam is presented. Using the Lagrange method, one can obtain the equations of motion for the system. The system equations will be linearized around $\theta_1 = \theta_2$.

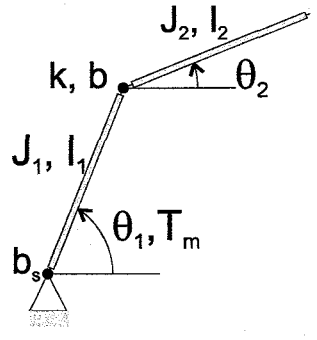

Figure 4.1: *Schematic view of the 2^{nd} order non-minimum phase system*

In this example we will control θ_2 by applying a torque T_m at θ_1. The transfer function then becomes (see appendix A.2):

$$\frac{\theta_2}{T_m} = 4\frac{-\left(\frac{1}{2}m_2 l_1 l_2\right)s^2 + bs + k}{A_4 s^4 + A_3 s^3 + A_2 s^2 + A_1 s} \tag{4.1}$$

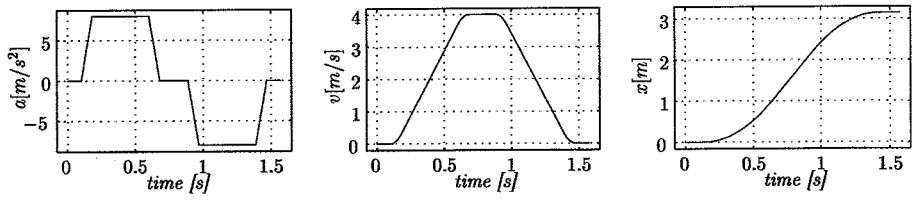

Figure 4.2: *Reference signal of the 4^{th} order system.*

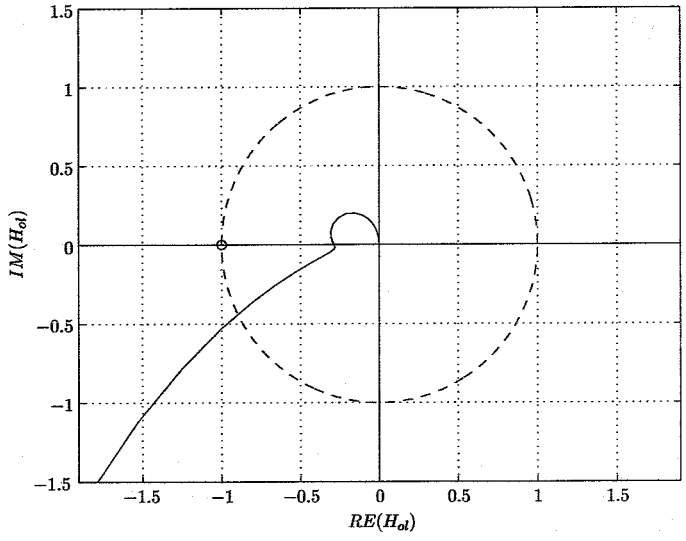

Figure 4.3: *Nyquist diagram of the open loop dynamics.*

Where,

$$\begin{aligned} A_1 &= b_m k \\ A_2 &= kJ_1 + \tfrac{1}{4}km_1l_1^2 + bb_m + kJ_2 + \tfrac{1}{4}km_2l_2^2 + m_2l_1l_2k \\ A_3 &= bJ_1 + \tfrac{1}{4}bm_1l_1^2 + b_mJ_2 + \tfrac{1}{4}b_mm_2l_2^2 + bJ_2 + \tfrac{1}{4}bm_2l_2^2 + m_2l_1l_2b \\ A_4 &= J_1J_2 + \tfrac{1}{4}J_1m_2l_2^2 + \tfrac{1}{4}m_1l_1^2J_2 + \tfrac{1}{16}m_1l_1^2m_2l_2^2 - \tfrac{1}{4}m_2^2l_1^2l_2^2 \end{aligned} \tag{4.2}$$

Figure 4.2 shows the reference position, velocity and acceleration signals. The system is controlled with a soft PD-controller, because the goal of this example is to show the strength of the learning algorithm. The open loop dynamics of the controlled system is presented in figure 4.3.

When applying a zero feedforward, the resulting tracking error is as presented in figure 4.4. This tracking error and the knowledge of the system is used to calculate the new feedforward.

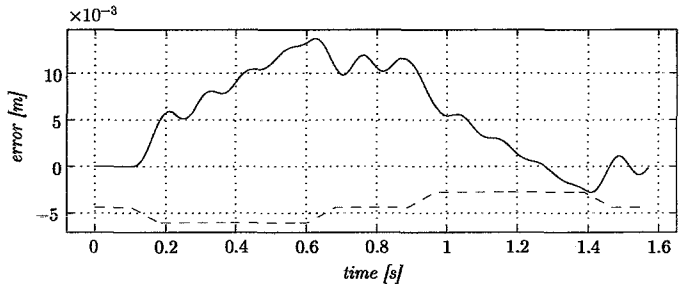

Figure 4.4: *Tracking error for the 4^{th} order system without using feedforward. The dashed line represents the reference acceleration signal.*

4.2 ILC applied to the non-minimum phase system

Now, we will use the two methods (ZPETC and D&F) to calculate a feedforward signal which minimizes the servo error for the setpoint in figure 4.2.

The ZPETC-method uses a robustness filter, Q. The choice of this filter is not straightforward, a number of different filter types can give good results. Here a 4^{th} order Butterworth low-pass filter is used. The choice of the cut-off frequency has a significant influence on the performance of the learning process. Here it is chosen such that the convergence criterion of equation 2.1 will be met amply.

Unlike the ZPETC method, the new D&F method for non-minimum phase systems allows the user to define begin and end conditions of the feedforward signal:

$$\underline{f}(t_0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \text{and} \quad \underline{f}(t_1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \text{, with} \quad \underline{f}(t) = \begin{bmatrix} f(t) \\ \dot{f}(t) \end{bmatrix} \tag{4.3}$$

In other words, the begin and end values of the feedforward, as well as the begin and end values of its first derivative must be zero.

Figure 4.5 shows the summated absolute error for the D&F as well as the ZPETC method. We can see that, in contradiction to the ILC problem for the stable fourth order system, now the new D&F method scores better than the ZPETC method.

Figure 4.6 shows the servo error after the first and the second iteration. After three iterations, a feedforward signal resulting in the smallest IAE has been found. The calculation times for the D&F method are significantly longer than the ZPETC method because of the combined stable/unstable poles in the learning filter. Using the same Q-filter from the ZPETC method for the feedforward update of the D&F method results in smaller calculation times, but the IAE is comparable to the ZPETC method.

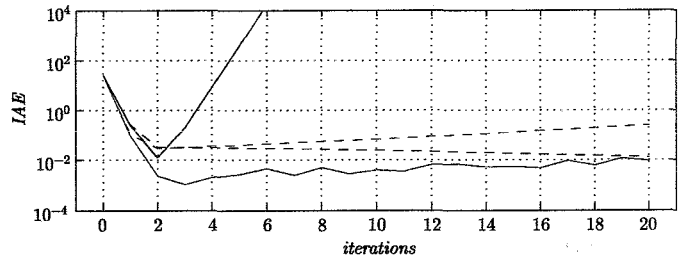

Figure 4.5: *Integrated Absolute Error for the non-minimum phase system using the ZPETC method (blue) and the D&F method (red). Without Q-filter (solid) and with Q-filter (dashed).*

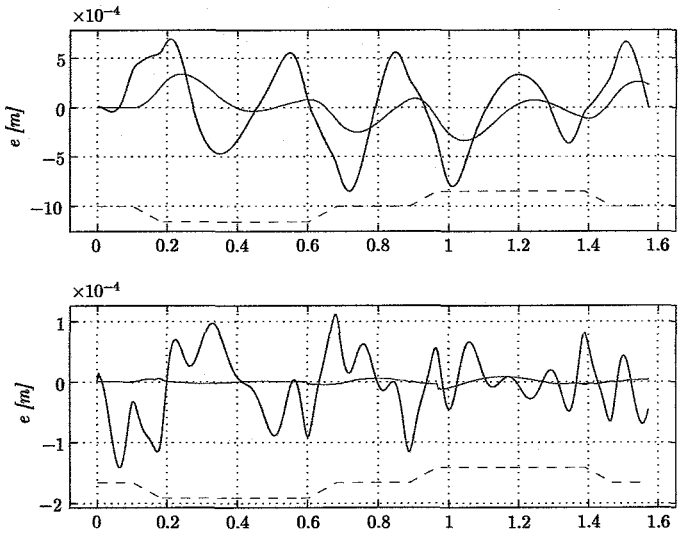

Figure 4.6: *Servo error after the first (top) and second (bottom) iteration for the non-minimum phase system using the ZPETC method (blue) and the D&F method (red).*

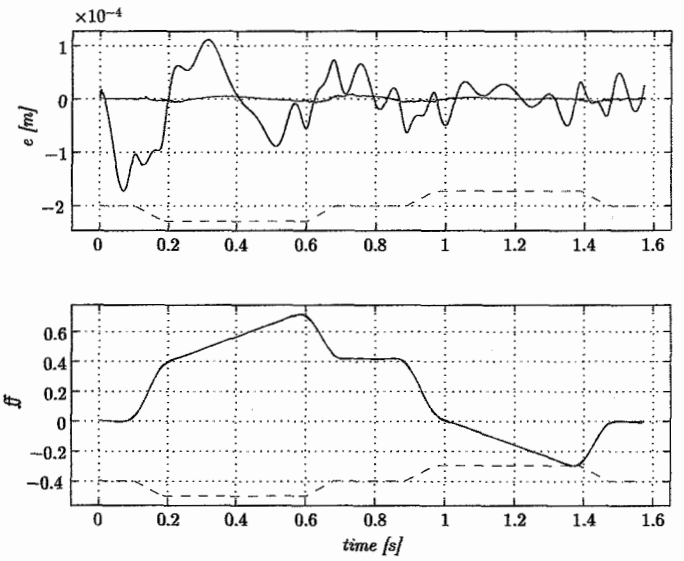

Figure 4.7: *Servo error (top) and feedforward signal (bottomt) for the non-minimum phase system using the ZPETC method (blue) and the D&F method (red).*

The servo error as well as the feedforward signal for both ILC methods after the third iteration signals are shown in figure 4.7.

Chapter 5

Conclusions and recommendations

5.1 Conclusions

The D&F method calculates the exact inverse of the process sensitivity, instead of the approximation calculated with the ZPETC-method. This implies that the D&F method will always perform optimal for each type of ILC problem. However, due to discretization the properties of the inverse process sensitivity will change slightly and the exactness of the method is lost. The differentiating filter, which is used in case of a non-causal part of the learning filter, will lead to imperfections around the beginning and the end of the signal.

The number of iterations needed to come to the most optimal feedforward signal are about the same for the two methods discussed in this report. For non-minimum phase systems, the D&F method will result in significantly better servo behavior.

There is no real need for a robustness filter after calculating the new feedforward with the D&F method. When the Q-filter is used, calculation times will drop, but the minimal IAE will be larger than the minimal IAE value without using a robustness filter.

The D&F method is, in terms of calculation time, not competitive to the ZPETC-method, when solving ILC problems for non-minimum phase systems. However, because of the offline calculation this weak point might not be a big problem.

With this D&F method, begin and end conditions for the feedforward can be demanded, so it will be suitable for designing repetitive tasks which are part of an array of different motions.

5.2 Recommendations

In future work, the D&F method has to be tested on an experimental set-up to check it's usability.

The calculation time of the RICLBVP-solver, used in this report, is very large. Maybe, the solver can improved to decrease the calculation time.

For stable systems, a two-point boundary solver has to be added to the MatLab tool, enabling demands for begin and end conditions for these systems too.

Bibliography

[1] M.J.G. van de Molengraft M. Steinbuch. Iterative Learning Control of Industrial Motion Systems. Eindhoven, University of Technology, Faculty of Mechanical Engineering, Systems and Control Group.

[2] M. Steinbuch M.G.E. Schneiders, M.J.G. van de Molengraft. Evaluation of (unstable) non-causal systems applied to iterative learning control. Eindhoven, University of Technology, Faculty of Mechanical Engineering, Systems and Control Group. DCT report 2001.08.

[3] B. Roset. Two-point boudary solver for stiff unstable linear systems (suited for application to ILC theory. Eindhoven, University of Technology, Faculty of Mechanical Engineering, Systems and Control Group. DCT report 2001.50.

Appendix A

Used systems

A.1 Stable 4^{th} order system

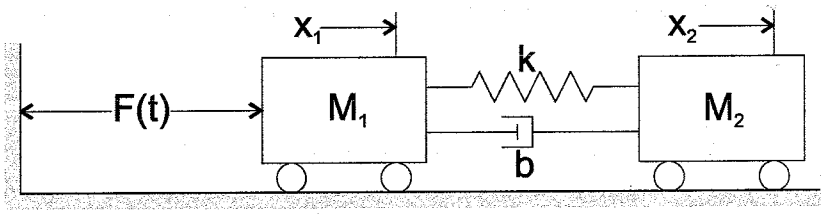

Figure A.1: *Schematic view of the 4^{th} order system*

$$\begin{cases} m_1\ddot{x}_1 = +(x_2 - x_1)\,k + (\dot{x}_1 - \dot{x}_2)\,b + F(t) \\ m_2\ddot{x}_2 = -(x_2 - x_1)\,k - (\dot{x}_1 - \dot{x}_2)\,b \end{cases} \tag{A.1}$$

After Laplace transformation:

$$\begin{aligned} \left(m_1 s^2 + bs + k\right) X_1(s) &= (bs + k)\,X_2(s) + F(s) \\ \left(m_2 s^2 + bs + k\right) X_2(s) &= (bs + k)\,X_1(s) \end{aligned} \tag{A.2}$$

Now, the two transfer functions become:

$$\begin{aligned} \frac{X_1}{F} &= \frac{m_2 s^2 + bs + k}{m_1 m_2 s^4 + b\,(m_1 + m_2)\,s^3 + k\,(m_1 + m_2)\,s^2} \\ \frac{X_2}{F} &= \frac{bs + k}{m_1 m_2 s^4 + b\,(m_1 + m_2)\,s^3 + k\,(m_1 + m_2)\,s^2} \end{aligned} \tag{A.3}$$

In this report, we will only look at the second transfer function, from $F \rightarrow x_2$.

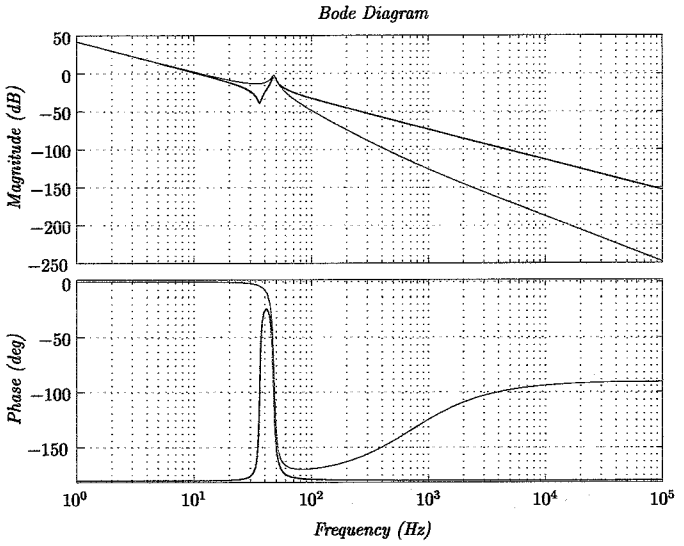

Figure A.2: *Bode diagram of the fourth order system. From $F \rightarrow x_1$ (blue) and $F \rightarrow x_2$ (green).*

The system parameters which are used during the simulations of this system are:

parameter	value	units
m_1	1.18×10^{-4}	kg
m_2	8.78×10^{-5}	kg
k	4.54	N/m
b	1.03×10^{-3}	N/ms

The bode diagram of the fourth order system is presented in A.2.

A.2 Non-minimum phase model

The Lagrange method will be used to obtain the equations of motion.

From the figure above, the following equations can be written for the position of the center of gravity for each link:

$$\begin{aligned} x_1 &= \tfrac{1}{2} l_1 \cos\theta_1 \\ y_1 &= \tfrac{1}{2} l_1 \sin\theta_1 \end{aligned} \tag{A.4}$$

$$\begin{aligned} x_2 &= l_1 \cos\theta_1 + \tfrac{1}{2} l_2 \cos\theta_2 \\ y_2 &= l_1 \sin\theta_1 + \tfrac{1}{2} l_2 \sin\theta_2 \end{aligned} \tag{A.5}$$

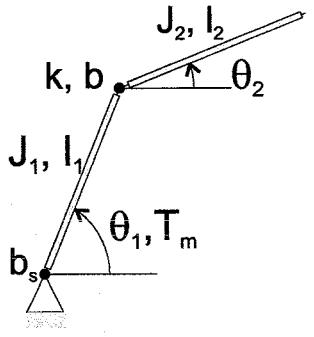

Figure A.3: *Schematic view of the 2^{nd} order non-minimum phase system*

And the velocities will respectively be:

$$\begin{aligned} \dot{x}_1 &= -\tfrac{1}{2} l_1 \dot{\theta}_1 \sin\theta_1 \\ \dot{y}_1 &= \tfrac{1}{2} l_1 \dot{\theta}_1 \cos\theta_1 \end{aligned} \tag{A.6}$$

$$\begin{aligned} \dot{x}_2 &= -l_1 \dot{\theta}_1 \sin\theta_1 - \tfrac{1}{2} l_2 \dot{\theta}_2 \sin\theta_2 \\ \dot{y}_2 &= l_1 \dot{\theta}_1 \cos\theta_1 + \tfrac{1}{2} l_2 \dot{\theta}_2 \cos\theta_2 \end{aligned} \tag{A.7}$$

$$\begin{aligned} \dot{z}_1 &= \tfrac{1}{4} {l_1}^2 {\dot{\theta}_1}^2 \\ \dot{z}_2 &= {l_1}^2 {\dot{\theta}_1}^2 + \tfrac{1}{4} {l_2}^2 {\dot{\theta}_2}^2 + l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_1 - \theta_2) \end{aligned} \tag{A.8}$$

Now that the velocities are known, the kinetic and potential energies can be calculated:

$$\begin{aligned} V &= \tfrac{1}{2} k (\theta_1 - \theta_2)^2 \\ T &= \tfrac{1}{2}(J_1 + \tfrac{1}{4} m_1 l_1^2)\dot{\theta}_1^2 + \tfrac{1}{2}(J_2 + \tfrac{1}{4} m_2 l_2^2)\dot{\theta}_2^2 + \tfrac{1}{2} m_2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_1 - \theta_2) \end{aligned} \tag{A.9}$$

Also the extra forces (acting on the center of gravity of each link) on the system can be written:

$$\mathbf{Q}^* = \begin{bmatrix} T_m - b_m \dot{\theta}_1 - b(\dot{\theta}_1 - \dot{\theta}_2) \\ -b(\dot{\theta}_2 - \dot{\theta}_1) \end{bmatrix} \tag{A.10}$$

According to Lagrange the equations of motion must obey:

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{q}} - \frac{\partial T}{\partial q} + \frac{\partial V}{\partial q} = Q^* \tag{A.11}$$

Eventually, the equation of motion for the first link is:

$$\begin{aligned} (J_1 + \frac{1}{4} m_1 l_1^2)\ddot{\theta}_1 \frac{1}{2} m_2 l_1 l_2 \ddot{\theta}_2 \cos(\theta_1 - \theta_2) + \frac{1}{2} m_2 l_1 l_2 \dot{\theta}_1^2 \sin(\theta_1 - \theta_2) + \ldots \\ \ldots + k(\theta_1 - \theta_2) + b_m \dot{\theta}_1 + b(\dot{\theta}_1 - \dot{\theta}_2) = T_m \end{aligned} \tag{A.12}$$

And for the second:

$$(J_2 + \frac{1}{4}m_2l_2^2)\ddot{\theta}_2 + \frac{1}{2}m_2l_1l_2\ddot{\theta}_1\cos(\theta_1 - \theta_2) - \frac{1}{2}m_2l_1l_2\dot{\theta}_1^2\sin(\theta_1 - \theta_2) + \\ ... + k(\theta_2 - \theta_1) + b(\dot{\theta}_2 - \dot{\theta}_1) = 0 \quad \text{(A.13)}$$

One can clearly see that these equations are nonlinear. The ILC method discussed in this report can only handle linear systems, so we will linearize the system around $\theta_1 = \theta_2$. The equations will now change to:

$$\begin{array}{l}(J_1 + \frac{1}{4}m_1l_1^2)\ddot{\theta}_1 + \frac{1}{2}m_2l_1l_2\ddot{\theta}_2 + k(\theta_1 - \theta_2) + b_m\dot{\theta}_1 + b(\dot{\theta}_1 - \dot{\theta}_2) = T_m \\ (J_2 + \frac{1}{4}m_2l_2^2)\ddot{\theta}_2 + \frac{1}{2}m_2l_1l_2\ddot{\theta}_1 + k(\theta_2 - \theta_1) + b(\dot{\theta}_2 - \dot{\theta}_1) = 0\end{array} \quad \text{(A.14)}$$

In matrix notation:

$$\begin{bmatrix} J_1 + \frac{1}{4}m_1l_1^2 & \frac{1}{2}m_2l_1l_2 \\ \frac{1}{2}m_2l_1l_2 & J_2 + \frac{1}{4}m_2l_2^2 \end{bmatrix}\begin{pmatrix}\ddot{\theta}_1 \\ \ddot{\theta}_2\end{pmatrix} + \begin{bmatrix} b_m + b & -b \\ -b & b\end{bmatrix}\begin{pmatrix}\dot{\theta}_1 \\ \dot{\theta}_2\end{pmatrix} + ... \\ ... + \begin{bmatrix} k & -k \\ -k & k\end{bmatrix}\begin{pmatrix}\theta_1 \\ \theta_2\end{pmatrix} = \begin{bmatrix} T_m \\ 0\end{bmatrix} \quad \text{(A.15)}$$

In short:

$$M\underline{\ddot{q}} + B\underline{\dot{q}} + K\underline{q} = \underline{u} \quad \text{(A.16)}$$

The transfer functions from $T_m \rightarrow \theta_1$ and $T_m \rightarrow \theta_2$ are:

$$\begin{array}{l}\frac{\theta_1}{T_m} = \frac{(J_2+\frac{1}{4}m_2l_2^2)s^2+bs+k}{A_4s^4+A_3s^3+A_2s^2+A_1s} \\ \frac{\theta_2}{T_m} = 4\frac{-(\frac{1}{2}m_2l_1l_2)s^2+bs+k}{A_4s^4+A_3s^3+A_2s^2+A_1s}\end{array} \quad \text{(A.17)}$$

Here,

$$\begin{array}{l}A_1 = b_mk \\ A_2 = kJ_1 + \frac{1}{4}km_1l_1^2 + bb_m + kJ_2 + \frac{1}{4}km_2l_2^2 + m_2l_1l_2k \\ A_3 = bJ_1 + \frac{1}{4}bm_1l_1^2 + b_mJ_2 + \frac{1}{4}b_mm_2l_2^2 + bJ_2 + \frac{1}{4}bm_2l_2^2 + m_2l_1l_2b \\ A_4 = J_1J_2 + \frac{1}{4}J_1m_2l_2^2 + \frac{1}{4}m_1l_1^2J_2 + 1/16m_1l_1^2m_2l_2^2 - \frac{1}{4}m_2^2l_1^2l_2^2\end{array} \quad \text{(A.18)}$$

The system parameters which are used during the simulations of this system are:

parameter	value	units
m_1	0.27	kg
m_2	0.27	kg
J_1	9.6×10^{-5}	$kg.m^2$
J_2	9.6×10^{-5}	$kg.m^2$
l_1	0.2	m
l_2	0.2	m
k	100	N/m
b	0.1	N/ms
b_m	1	N/ms

Figure A.4 shows the bode diagrams of the linearized non-minimum phase system.

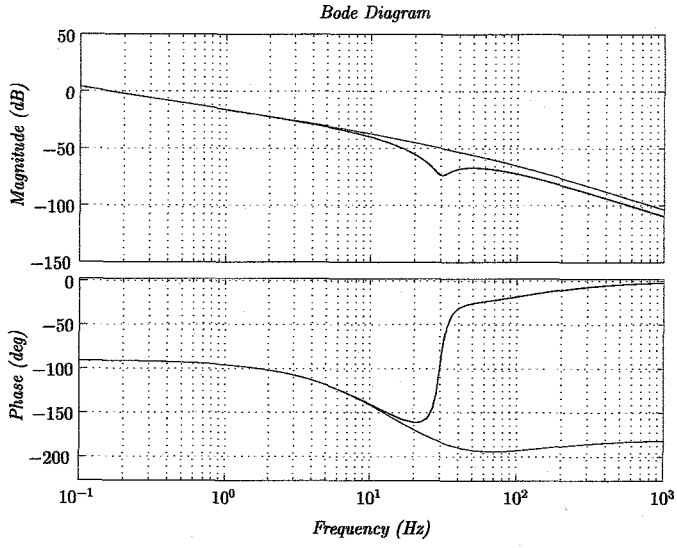

Figure A.4: *Bode diagram of the non-minimum phase system. From $T_m \rightarrow \theta_1$ (blue) and $T_m \rightarrow \theta_2$ (green).*

Appendix B

MATLAB implementation

B.1 Splitting the learning filter

```
function [Lc,Lnc]=SplitSystem(L);
% SplitSystem.m
% -------------
% This function splits a systemsinto a causal and a non-causal
% part. If the system has more zeros than poles, the order of the
% numerator is higher than that of the denumerator.
% The excess of zeros can be eliminated from the total system,
%  forming the non-causal part (Lnc). The causal system is the
%  system remaining from the original system minus the non-causal
% system. Example:
%         (s-z1)(s-z2)(s-z3)              (s-z2)(s-z3)
% L(s)= k ------------------ = k*(s-z1)*------------ = Lnc * Lc
%            (s-p1)(s-p2)                 (s-p1)(s-p2)
%
% If no non-causal or causal part exists, then respectively Lnc
% and Lc will be equal to 1. (Note: if both parts aren't equal to
% 1, then Lnc includes the system gain k).
%
% usage: [Lc,Lnc] = SplitSystem(L)
% where:     L    : System to split.
%            Lc   : Causal part of L.
%            Lnc  : Non-causal part of L.

% Author : Arjan Teerhuis, 2001

if not(isobject(L))
   error('System L is not a lti system')
end
```

```
% ----------------------------------
% are there are more zeros than poles
% ----------------------------------
[zz,pp,kk]=zpkdata(L,'v');
nd=length(zz)-length(pp);
if nd>0
   % more zeros than poles, so differentiating action is needed
   % seperate zeros (keep compl. conj. pares together)
   % fill two vectors with real poles and compl. conj. pares
   N_real=[]; N_conj=[];
   for i=1:length(zz)
      if imag(zz(i))==0
         % puur reeel
         N_real=[N_real;zz(i)];
      else
         N_conj=[N_conj;zz(i)];
      end
   end
   % check if causal part can be filled with conj. pares
   % perhaps, one or more real poles must be added
   if length(pp)>0
      % causal filter needed!
      if floor(length(pp)/2)==length(pp)/2
         % even number of poles
         if length(N_conj)<=length(pp)
            % all conj. pares can be used
            zz=N_conj;
            n=length(pp)-length(zz);
            if n>0
               % room left for real poles
               zz=[zz;N_real(1:n)];
               dd=N_real(n+1:end);
            else
               dd=N_real;
            end
         else
            % only a few conj. pares can be used
            zz=N_conj(1:length(pp));
            if length(N_real)>0
               dd=[N_conj(length(pp)+1:end);N_real];
            else
               dd=[N_conj(length(pp)+1:end)];
            end
         end
      else
         % odd number of poles
```

```
            if length(N_conj)<=2*floor(length(pp)/2)
               % all conj. pares can be used + one real
               zz=[N_conj;N_real(1)];
               dd=N_real(2:end);
            else
               % only a few conj. pares can be used + real
               if length(N_real)>0
                  % room left for real pole in causal filter
                  zz=[N_conj(1:2*floor(length(pp)/2));N_real(1)];
                  if length(N_real)>1
                     % room left for real numbers in diff action
                     dd=[N_conj(2*floor(length(pp)/2)+1:end)]
                     dd=[dd;N_real(2:end)];
                  else
                     dd=[N_conj(2*floor(length(pp)/2)+1:end)];
                  end
               else
                  zz=[N_conj(1:2*floor(length(pp)/2))];
                  dd=[N_conj(2*floor(length(pp)/2)+1:end)];
               end
            end
         end
      else
         % no causal filter needed, just a diff. filter
         zz=[];
         dd=[N_real;N_conj];
      end
      Lnc=zpk(dd,[],kk);
      Lc=zpk(zz,pp,1);
   else
      % no non-causal filter needed
      dd=[];
      Lnc=zpk([],[],1);
      Lc=zpk(zz,pp,kk);
   end
```

B.2 Using the differentiating filter(s)

```
[numDD,denDD]=tfdata(Lnc,'v');
DiffOrde=length(numDD)-1;
DiffSigs=[];
if DiffOrde>0
   DiffSigs=e(:);
   Ync=numDD(DiffOrde+1)*e(:);
   for Orde=1:DiffOrde
      CC=bdiffilt(Orde,2*Orde,'PInt',2*Orde);
      DiffSig=diffilt(e(:),CC,Orde,dt);
      DiffSigs=[DiffSigs DiffSig(:)];
      Ync=Ync+numDD(DiffOrde+1-Orde)*DiffSig;
   end
else
   Ync=e(:);
end
DiffSigs=[DiffSigs Ync(:)];
```

B.3 Filtering the unstable system

When the error signal has been filtered by the non-causal filter, it has to be fed through the causal filter. Because of the example used in this report, this filter has unstable poles. To solve this problem, the RICLBVP-solver is used. Before the causal filter can be presented to it, the upper part of the state-space notation must contain the stable equations (the first k equations).

```
[NUM_Lc,DEN_Lc]=tfdata(Lc,'v');
if not(length(NUM_Lc)==length(DEN_Lc))
   NUM_Lc=[zeros(1,length(DEN_Lc)-length(NUM_Lc)) NUM_Lc];
end
AA=[-DEN_Lc(2:end)' ...
       ... [eye(length(DEN_Lc)-2);zeros(1,length(DEN_Lc)-2)]];
BB=NUM_Lc(2:end)'-NUM_Lc(1).*DEN_Lc(2:end)';
CC=[1 zeros(1,size(AA,1)-1)];
DD=NUM_Lc(1);
[AA,BB,CC,DD,k,order]=sysreorder(AA,BB,CC,DD);
[Ma,Mb,d0]=boundvals(CC,DD,wa,wb,d,Ync);
X=riclbvp(AA,BB,Ync,t,Ma,Mb,d0,k,Rmax,RelTol,AbsTol, ...
       ... interp_methode,ode);
uu=CC*X+DD*Ync(:)';
```

Appendix C

Rewriting the Learning Filter in State-Space notation

This section shows how the initial and the end conditions for the state can be calculated in the case of any demands in derivatives of the feedforward. The solver used here, asks for a state-space representation of the causal part of the learning filter:

$$L_c(s) \rightarrow \begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases} \tag{C.1}$$

To rewrite the num/den representation into a state-space notation, the observable canonical form is used:

$$A = \begin{bmatrix} -a_{K-1} & 1 & & 0 \\ \vdots & & \ddots & \\ -a_1 & 0 & & 1 \\ -a_0 & 0 & \dots & 0 \end{bmatrix}, B = \begin{bmatrix} b_{K-1} \\ \vdots \\ b_1 \\ b_0 \end{bmatrix} - b_K \begin{bmatrix} a_{K-1} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix} \tag{C.2}$$

$$C = \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix} \qquad , D = b_K$$

To calculate any derivatives of $y(t)$, two more matrices must be known:

$$y(t) = Ox(t) + Ru(t) \tag{C.3}$$

where:

$$y(t) = \begin{bmatrix} y(t) \\ \dot{y}(t) \\ \vdots \\ y^{K-1}(t) \end{bmatrix}, x(t) = \begin{bmatrix} x(t) \\ \dot{x}(t) \\ \vdots \\ x^{K-1}(t) \end{bmatrix}, u(t) = \begin{bmatrix} u(t) \\ \dot{u}(t) \\ \vdots \\ u^{K-1}(t) \end{bmatrix}$$

The matrices O and R are defined like:

$$R = \begin{bmatrix} D & 0 & 0 & 0 & 0 & \dots & 0 \\ CB & D & 0 & 0 & 0 & \dots & 0 \\ CAB & CB & D & 0 & 0 & \dots & 0 \\ CA^2B & CAB & CB & D & 0 & \dots & 0 \\ & & & \vdots & & & \\ CA^{K-2}B & CA^{K-3}B & \dots & \dots & CAB & CB & D \end{bmatrix} \qquad \text{(C.4)}$$

$$O = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{K-1} \end{bmatrix}$$

From here, the calculation of the boundary matrices B_a, B_b and d is the same as equation (3.2), but now the C and D matrices are replaced by the O and R matrices respectively.