Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

Bakalářská práce

PLZEŇ 2021

Jan Trejbal

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne 16.8.2021 .....

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jan TREJBAL**
Osobní číslo:	**A18B0550P**
Studijní program:	**B3918 Aplikované vědy a informatika**
Studijní obor:	**Kybernetika a řídicí technika**
Téma práce:	**Metody Iterative learning control pro mechatronické systémy**
Zadávající katedra:	**Katedra kybernetiky**

Zásady pro vypracování

1. Seznamte se s technikami Iterative learning control a způsoby jejich použití v systémech řízení pohybu.
2. Implementujte a otestujte vybrané metody v simulačním prostředí systému Matlab.
3. Implementujte metody ILC do prostředí reálného času.
4. Otestujte implementované metody na vhodném mechatronickém systému a proveďte srovnání dosažitelné kvality řízení ve srovnání se standardní zpětnovazební regulací.

Rozsah bakalářské práce: **30-40 stránek A4**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**
Jazyk zpracování: **Angličtina**

Seznam doporučené literatury:

D.A. Bristow, M. Tharayil and A.G. Alleyne, „A Survey of Iterative Learning Control: A Learning-Based Method for High-Performance Tracking Control",2006
Bien, Xu, „Iterative Learning Control – analysis, design, integration and applications", Kluwer, 1998

Vedoucí bakalářské práce: **Ing. Martin Goubej, Ph.D.**
Katedra kybernetiky

Datum zadání bakalářské práce: **15. října 2020**
Termín odevzdání bakalářské práce: **24. května 2021**

Západočeská univerzita v Plzni
L.S.
Fakulta aplikovaných věd

Doc. Dr. Ing. Vlasta Radová
děkanka

Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

V Plzni dne 15. října 2020

Abstrakt Tato práce se zabývá návrhem a implementací metod Iterative Learning Control pro zlepšení výkonnosti systémů řízení pohybu. V úvodní části jsou představeny metody Iterative Learning Control a je řešena otázka jejich konvergence. Druhá část se zabývá implementací a testováním jednotlivých metod ILC v prostředí Matlab. V této části je také zkoumán vliv konstant jednotlivých metod ILC na jejich chování. Třetí část se zabývá implementací metod ILC v prostředí reálného času. Čtvrtá část popisuje implementaci metod ILC pro řízení reálného systému. Čtvrtá část také obsahuje výsledky experimentů s reálným systémem a diskusi k výsledkům.

Klíčová slova: Iterativní učící se řízení, řízení pohybu, mechatronika, elektrické pohony

Abstract This thesis deals with the design and implementation of Iterative Learning Control methods to improve the performance of motion control systems. The beginning part introduces Iterative Learning Control methods and addresses the question of their convergence. The second part covers the implementation and testing of individual ILC methods in Matlab. In this part, the effects of the constants of the different ILC methods on their behaviour are also examined. The third part deals with the implementation of ILC methods in a real-time environment. The fourth part describes the implementation of ILC methods for real system control. The fourth section also contains the results of experiments with a real system and a discussion of the results.

Keywords: Iterative Learning Control, motion control, mechatronics, electrical drives

List of Figures

1.1 Control scheme . . . 4
1.2 Comparison of simulated error without and with ILC . . . 5
1.3 Mixed time/task-domain block diagram of PD ILC . . . 6
1.4 Mixed time/task-domain block diagram of FD ILC . . . 7
1.5 Mixed time/task-domain block diagram of BF ILC with constant Ψ . . . 11
1.6 Mixed time/task-domain block diagram of BF ILC with continuously calculated Ψ . . . 12

2.1 System regulated by PI regulator . . . 15
2.2 Model of system . . . 16
2.3 System regulated by PID regulator . . . 17
2.4 Simulation scheme . . . 18
2.5 Calculated trajectories . . . 18
2.6 Simulation scheme . . . 19
2.7 RMS based on k_v and k_a . . . 19
2.8 Comparison of combined and feedback control . . . 20
2.9 Simulation scheme ILC . . . 20
2.10 Comparing different values of k_p, k_d and bandwidth . . . 21
2.11 Bode plot of GSL . . . 22
2.12 Comparing RMS with different α . . . 23
2.13 BF ILC with matrices calculated by the common method . . . 23
2.14 Comparing RMS with different α . . . 24
2.15 Results of simulation with $\alpha = 0.1$ . . . 25
2.16 Comparison of different ILC variants, stable reference trajectories . . . 25
2.17 Comparison of different ILC variants when the reference trajectory changes 26

3.1 Implementation in REXYGEN . . . 27
3.2 Implementation in REXYGEN fast-task PD and FD ILC . . . 28
3.3 Implementation in REXYGEN fast-task BF . . . 29
3.4 States and transition of ATMT . . . 30
3.5 Transposed direct form 2 implementation of a second order filter [15] . . . 31
3.6 Comparison of results in Matlab and Rexygen PD ILC . . . 31
3.7 Comparison of results in Matlab and Rexygen BF ILC . . . 32
3.8 Comparison of results in Matlab and Rexygen FD ILC . . . 32

4.1 Controlled system . . . 33
4.2 Comparison of similarities of identified transfer functions with 3, 4, 5 zeros against measured data. . . . 34
4.3 Identified system regulated by PID regulator . . . 35
4.4 Comparison of real-system control results using different variants of ILC . 36
4.5 Response to trajectory change using different variants of ILC . . . 37
4.6 ILC control after the convergence is completed for different variants of ILC 38

Contents

Used symbols and abbreviations **2**

1 Theory of Iterative Learning Control **4**
1.1 PD-ILC . . . 5
1.2 FD-ILC . . . 7
1.2.1 ZPTEC . . . 8
1.3 Convergence of PD and FD ILC . . . 9
1.3.1 With Q = 1 . . . 9
1.3.2 With general Q . . . 10
1.3.3 Zero phase digital filtering . . . 10
1.3.4 Testing convergence of PD-ILC . . . 10
1.4 BF-ILC . . . 11
1.4.1 Calculating Q and L . . . 12

2 Simulations in Matlab **14**
2.1 Model used in simulations . . . 14
2.1.1 Creating model of controlled system . . . 14
2.1.2 Designing PID regulator . . . 16
2.2 Generating reference signal . . . 17
2.3 Feedforward control . . . 18
2.3.1 Finding optimal values of constants . . . 19
2.3.2 Comparing combined and feedback - only control . . . 19
2.4 Simulating ILC in Matlab . . . 20
2.4.1 Simulating PD-ILC . . . 20
2.4.2 Simulating FD-ILC . . . 22
2.4.3 Simulating BF-ILC . . . 23
2.4.4 Comparison of different ILC . . . 25

3 Implementation in Rexygen **27**
3.1 Introduction of Rexygen . . . 27
3.2 Implementation of ILC in Rexygen . . . 27
3.2.1 Scheme of implementation . . . 27
3.2.2 Program runtime control using ATMT . . . 29
3.2.3 Python block . . . 30
3.3 Implementing ILC with PYTHON block . . . 30
3.3.1 IIR filter . . . 31
3.3.2 FIR filter . . . 31
3.4 Comparing implementation in Rexygen with Maltab . . . 31

4 Controlling real mechatronic system **33**
4.1 Description of the controlled system . . . 33
4.2 System identification using measured data . . . 33
4.3 Design of a PID controller to control the identified system. . . . 34
4.4 ILC implementation for real - system control . . . 35
4.5 Data measured on a real - system . . . 36

5 Conclusion **39**

Used symbols and abbreviations

Ψ Basis functions

Θ Vector of coefficients

α Learning constant

$e_j(k)$ k th element of vector of errors in j th iteration

$u_j(k)$ k k th element of feedforward input in j th iteration

D unit circle

I Identity matrix

N Length of the reference signal r

k_p Learning constant

k_p Learning constant

m Number of basis functions

C PID regulator

G Controlled system

ILC Iterative Learning Control

L Learning function (PD ILC) / Filter (FD ILC) / Learning matrix (BF ILC)

Q Filter (PD and FD ILC) / Robustness matrix (BF ILC)

r A reference trajectory

RMS Root Mean Square

RMSE Root Mean Square Error

S Sensitivity function $\frac{1}{1+CG}$

1. Theory of Iterative Learning Control

If the feedback-controlled system performs the task repeatedly, the error is very similar, as shown in Figure 1.2.a . Thus, there is no improvement and the control performance is constant.

The main idea of Iterative Learning Control is to use the fact that a controlled system will execute the same task multiple times. With the error information from past tasks, we can predict an error in a future task and try to compensate for it. [13, 9]
ILC can be summarized as the repeated execution of the following steps:

1. The result of the last execution of task is measured and the error is calculated.
2. Based on the calculated error, a new feedforward control is calculated.
3. The computed feedforward control is fed into the system the next time the task is executed.
4. The task is executed.

The calculation of the feedforward signal after each iteration is usually done not only based on the error in the last iteration, but also based on the feedforward signal in the last iteration. Different ways of calculating feedforward signal will be discussed in the following sections The feed-forward signal is stored in memory and fed as f into the system during the task as described in Figure 1.1.

If we compare the performance of the system controlled by a PID controller and ILC, Figure 1.2.b , with the performance of the system controlled only by a PID controller , Figure 1.2.a , we can see a gradual improvement in the performance of the system with ILC. The improvement can also be seen by comparing the root mean square error in the iterations in Figure 1.2.c . The ILC performance stabilizes at asymptotic error after several iterations. This value will be non-zero due to simulated noise and other limitations. Due to ILC, the error is significantly reduced after several iterations.

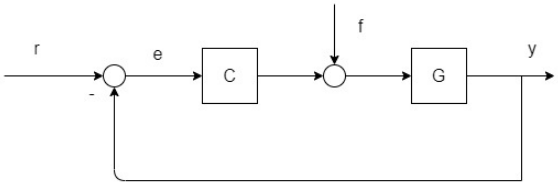
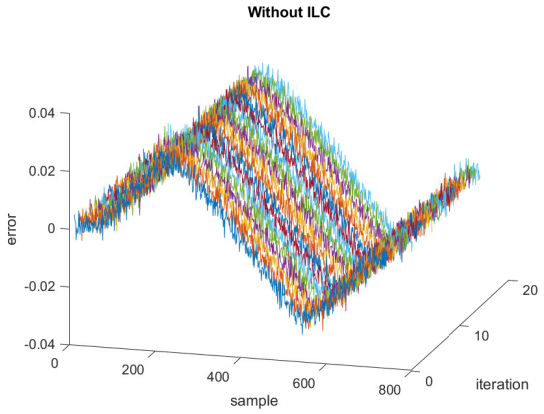


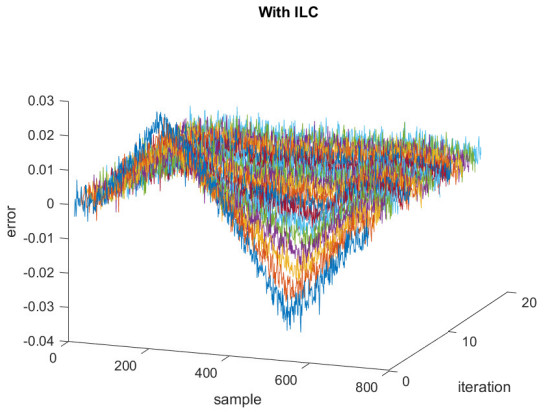
Figure 1.1: Control scheme

ILC is usually implemented together with a feedback controller, which helps with non-repeating disturbances. ILC is sensitive to nonrepeating disturbances and noise. Offline computation used in ILC allows advanced filtering, for example zero-phase filtering. Offline filtering reduces ILC sensitivity to noise. Due to working with data from the previous iteration, discrete time is the natural environment for ILC. ILC can significantly increase control performance compared to pure feedback control. ILC offers high-performance control after enough iterations. ILC is also highly robust to system uncertainties, if implemented correctly.

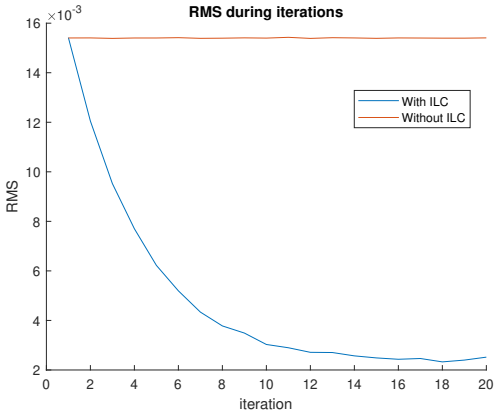
ILC can be used to improve the performance of systems that perform a repetitive task. This includes in particular systems in manufacturing, robotics and chemical processing. For example, ILC has already been successfully applied to industrial robots, injection-molding machines, semibatch chemical reactors and wafer stage motion systems. [9]



(a) System controlled only by PID controller



(b) System controlled by PID controller and ILC



(c) RMS error comparison

Figure 1.2: Comparison of simulated error without and with ILC

1.1 PD-ILC

The PD type of ILC is probably the most widely used type of learning function [9]. PD ILC learning function consists of proportional and derivation part. Both parts have an independent learning gain that can be manually tuned. The difference between $k+1$ and k element in the error vector is used to approximate derivation. [4, 10] PD learning function can be written as

$$u_{j+1}(k) = Q[u_j(k) + k_p e_j(k+1) + k_d(e_j(k+1) - e_j(k))]$$

Q is sometimes set as Q = 1, which means that the algorithm does not contain Q filtering. This choice is necessary for perfect tracking. On the other hand, choosing a non-unit Q filter can improve the robustness of the control algorithm and guarantee monotone convergence, as discussed in Section 1.3 . The value of constants k_p and k_d is usually selected by manual tuning. The Q filter is often chosen as a low pass filter to improve robustness and filtering of high frequency noise.

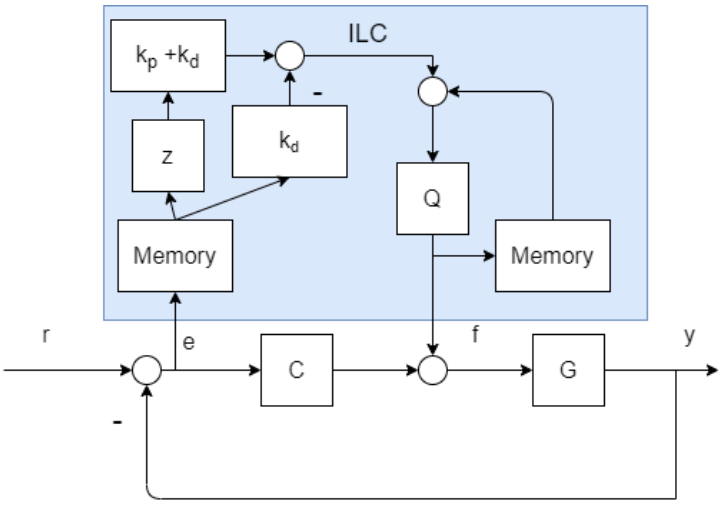


Figure 1.3: Mixed time/task-domain block diagram of PD ILC

One possible tuning approach is:

1. Choose the order and type of filter.
2. Manually tune the value of k_p, k_d and the bandwidth of the Q filter. Start with lower values and try to get stable transient behaviour.
3. After achieving stable behaviour, increase the value of k_p, k_d and the bandwidth of the Q filter for better performance. Repeat this until unstable behaviour is achieved. Then set the value of k_p, k_d and the bandwidth to the best performing and stable combination.

Another possible tuning approach is [9]:

1. Choose the order and type of filter.
2. Manually tune the value of k_p and k_d. Consider a Q filter with constant and low bandwidth. Choose value of k_p and k_d. Run a simulation for a sufficient number of iteration to determine transient behavior and asymptotic error. Choose the combination of k_p and k_d with the best performance.
3. Tune the bandwidth of the Q filter. Choose the highest possible bandwidth with stable transient behavior.

The value of k_p and k_d determines the convergence rate and asymptotic error; usually a higher value of k_p and k_d means faster convergence. The bandwidth of the Q filter for constant values of k_p and k_d have almost no effect on the convergence rate, but it determines the value of asymptotic error.

1.2 FD-ILC

The main idea of Frequency Domain ILC is to design L as an approximate inversion of GS. [16] Consider a learning function in the form of

$$u_{j+1} = Q(u_j + Le_j)$$

Then error propagation, as proven in 1.3 is

$$e_{j+1} = Q(1 - GSL)e_j + (1 - Q)Sr$$

If L was GS^{-1} and Q was 1, the error would be zero after one iteration. [13] This makes it desirable that $L \approx (GS)^{-1}$ and $Q \approx 1$. Zero phase digital filtering is used to prevent a phase shift of the Q filter. In order to avoid a large response to non-repetitive disturbances, the learning function can be modified to the form

$$u_{j+1} = Q(u_j + \alpha Le_j)$$

A low value of α causes longer convergence but reduces the response to non-recurrent disturbances and may lead to lower asymptotic error.

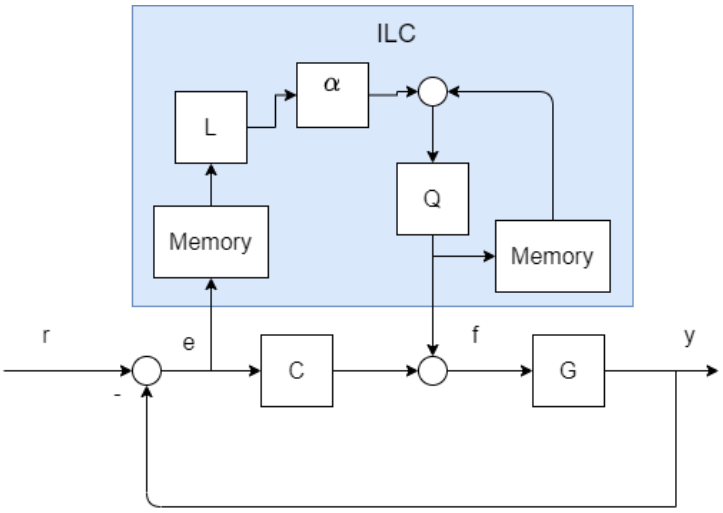


Figure 1.4: Mixed time/task-domain block diagram of FD ILC

One possible approach of implementing FD-ILC is:

1. Create a model of the system and calculate the sensitivity function S, then create L as $L \approx (GS)^{-1}$. For example by ZPTEC, as described in 1.2.1.

2. Design a low-pass Q filter. The Q filter should be designed based on (1-GSL) so that the convergence conditions described in 1.3 are satisfied. If zero phase digital filtering is used, then the convergence conditions described in section 1.3.3 have to be met.

1.2.1 ZPTEC

The Zero Phase Error Tracking Algorithm [17] produces a stable inversion of GS, called L, for which the LGS transfer has zero phase and at low frequencies, the gain is close to 1. The transfer function GS can be written as

$$GS(z^{-1}) = \frac{z^{-d}B_s(z^{-1})B_u(z^{-1})}{A(z^{-1})}$$

- z^{-d} delay
- $B_s(z^{-1})$ zeros strictly in D
- $B_u(z^{-1})$zeros outside D
- $A(z^{-1})$ poles

The unstable zeros of the GS transfer cannot be inverted to the L poles because unstable poles would be created. It is also not possible to omit the unstable zeros in the inversion, as this would cause a phase shift. To get rid of the phase shift created by the unstable zeros in GS, additional stable zeros $B_u(z)$ must be added to L. Then

$$L(z^{-1}) = \frac{z^{d}A(z^{-1})B_u(z)}{\beta B_s(z^{-1})}$$

Then LGS can be written as

$$LGS = \frac{B_u(z)B_u(z^{-1})}{\beta}$$

Zeros $B_u(z)$ are created by mapping unstable zeros $B_u(z^{-1})$ inside D. This is done by substituting z for z^{-1} in $B_u(z^{-1})$.
$B_u(z)B_u(z^{-1})$ can be analysed:

$$B_u(z)B_u(z^{-1}) = B_u(e^{j\omega})B_u(e^{-j\omega})$$

by using Euler's formula $e^{j\omega} = cos(\omega) + jsin(\omega)$

$$= [Re(\omega) - jIm(\omega))][Re(\omega) + jlm(\omega)] = Re(\omega)^2 + Im(\omega)^2$$

This expression does not have any imaginary part, which means that the phase shift will be zero.
β is chosen as $[B_u(1)]^2$ to create a gain close to 1 on low frequencies.
L can be then written as

$$L(z^{-1}) = \frac{z^{d}A(z^{-1})B_u(z)}{[B_u(1)]^2 B_s(z^{-1})}$$

Unstable zeros and delay in GS lead to non-causal L. Non causal L can be written in form

$$L = \underbrace{z^{d+s}}_{\text{time shift}} * \underbrace{\frac{z^{d}A(z^{-1})B_u^*(z^{-1})}{\beta B_s(z^{-1})}}_{\substack{L_c \\ \text{causal}}}$$

Filtering by the L filter can therefore be done as filtering a signal shifted by $d + s$ by the causal filter L_c.

1.3 Convergence of PD and FD ILC

Convergence of PD and FD ILC can be analysed to ensure theoretical convergence of implemented ILC control. The ILC algorithm with learning update

$$u_{j+1} = Q[u_j + Le_j]$$

was analysed. Q and L are filters that can be non-causal. First the ILC algorithm with $Q = 1$ was analysed. This choice should lead to high performance. Then the ILC algorithm with general Q was analysed. A correctly designed Q filter helps with robustness. [9, 16]

1.3.1 With Q = 1

The learning update

$$u_{j+1} = u_j + Le_j$$

The error in j iteration is given by

$$e_j = Sr - GSu_j$$

This equation can be modified to form

$$u_j = (GS)^{-1}(Sr - e_j)$$

The equation for error in iteration is valid for all iterations

$$e_{j+1} = Sr - GSu_{j+1}$$

Now we can substitute for u_{j+1} from the learning update

$$e_{j+1} = Sr - GSu_{j+1} = Sr - GS(u_j + Le_j)$$

and substitute for u_j from the modified form

$$e_{j+1} = Sr - GS[(GS)^{-1}(Sr - e_j) + Le_j]$$

The equation can be simplified

$$= Sr - (Sr - e_j) - GSLe_j = e_j - GLSe_j$$

to form

$$e_{j+1} = (1 - GSL)e_j$$

The error will therefore evolve according to the chosen filter L, the sensitivity function S and the system G.
If the expression $(1 - GSL)$ behaves like contraction mapping, then the error will converge monotonously to a certain value. To ensure contraction mapping the gain of $(1 - GSL)$ must be less then 1. [9, 6, 16] Contraction mapping for all signals can be tested as:

$$|(1 - G(e^{j\omega})S(e^{j\omega})L(e^{j\omega})| < 1 \quad \forall\omega \in [0, 2\pi]$$

This is the equivalent of a magnitude less then $0dB$ on all frequencies. [9, 16]

1.3.2 With general Q

The learning update

$$u_{j+1} = Q(u_j + Le_j)$$

The error in j iteration is given by

$$e_j = Sr - GSu_j$$

This equation can be modified to form

$$u_j = (GS)^{-1}(Sr - e_j)$$

The equation for error in iteration is valid for all iterations

$$e_{j+1} = Sr - GSu_{j+1}$$

Now we can substitute for u_{j+1} from the learning update

$$e_{j+1} = Sr - GSu_{j+1} = Sr - GSQ(u_j + Le_j)$$

and substitute for u_j from the modified form

$$e_{j+1} = Sr - QGS[(GS)^{-1}(Sr - e_j) + Le_j]$$

The equation can be simplified

$$= Sr - (QSr - Qe_j) - QGSLe_j = Qe_j - QGLSe_j$$

to form

$$e_{j+1} = Q(1 - GSL)e_j + (1 - Q)Sr$$

The error will therefore evolve according to the chosen Q filter, L filter, the sensitivity function S and the system G.
With $Q \neq 1$, the steady-state error for a non-zero reference signal will not be 0. A magnitude of Q can be equal to 1 for some frequencies, which allows perfect tracking.
If the expression $(1-GSL)$ behaves like contraction mapping, then the error will converge monotonously to a certain value. To ensure contraction mapping the gain of $(1 - GSL)$ must be less then 1. [9, 6, 16] Contraction mapping for all signals can be tested as:

$$|Q(e^{j\omega})(1 - G(e^{j\omega})S(e^{j\omega})L(e^{j\omega})| < 1 \quad \forall\omega \in [0, 2\pi]$$

This is the equivalent of a magnitude less then $0dB$ on all frequencies.

1.3.3 Zero phase digital filtering

If we want to use zero phase digital filtering to get rid of the phase shift, (Matlab's filterfilt command) we have to consider a condition [16] in the form:

$$|Q(e^{j\omega})|^2|(1 - G(e^{j\omega})S(e^{j\omega})L(e^{j\omega})| < 1 \quad \forall\omega \in [0, 2\pi]$$

1.3.4 Testing convergence of PD-ILC

The learning function L of PD-ILC can be considered as a discrete transfer function

$$L = (k_p + k_d)z - k_d$$

It is therefore possible to verify PD-ILC convergence as described above. Thus, the convergence of PD-ILC will depend on the choice of the constants k_p and k_d and the choice of the Q filter in the case of PD-ILC with a filter.

1.4 BF-ILC

The key assumption in PD and FD ILC is a constant reference during repetition. As a result the learned feedforward signal is only optimal for a specific task. Consequently, the extrapolation of feedforward signal to other tasks causes serious performance deterioration. [5, 7]

In BF-ILC extrapolation ability is increased by using basis function. BF-ILC does not learn the feedforward signal , but instead learns the vector of coefficients Θ, which when multiplied by the basis functions gives the feedforward signal. The ideal value of the vector of coefficients Θ usually depends mainly on the controlled system and is very similar for the whole class of reference signals. Examples of basis functions commonly used in BF ILC include $\dot{r}$, $\ddot{r}$ and $sign(r)$.

Θ is recalculated after each iteration according to the formula [5]

$$\theta_{j+1} = Q\theta_j + \alpha L e_j$$

where Q and L are the learning matrices that must be computed before learning begins using one of the methods described later.

Based on the recalculated Θ, the feed forward signal is then calculated as:

$$u_{j+1} = \Psi\theta_{j+1}$$

This calculation can be done for constant basis functions Ψ before the task starts - Figure 1.5 , then we know the feed forward signal for the whole task, or it can be done continuously during the task from continuously calculated values of the basis functions - figure 1.6.

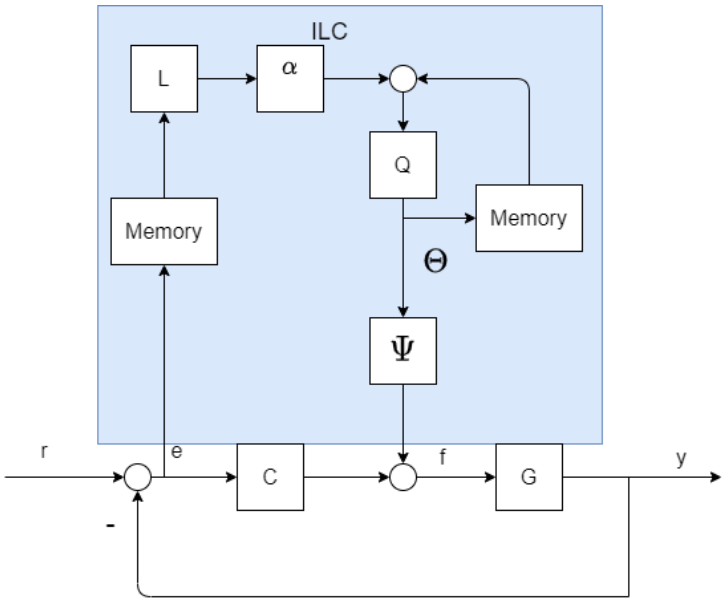


Figure 1.5: Mixed time/task-domain block diagram of BF ILC with constant Ψ

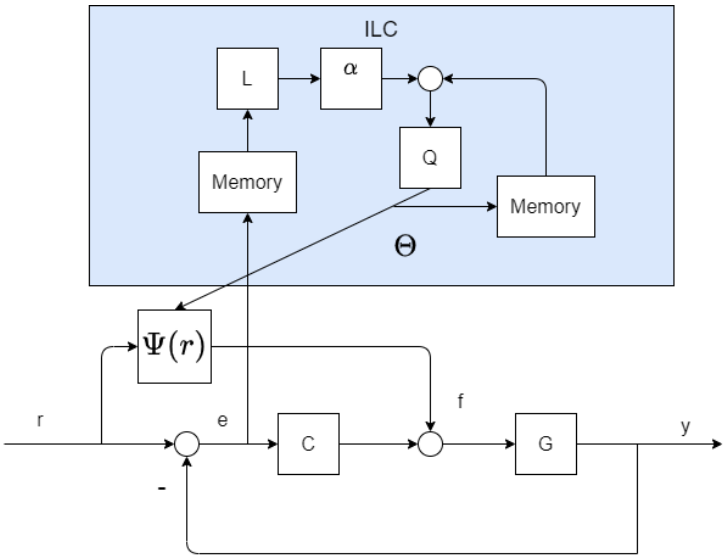


Figure 1.6: Mixed time/task-domain block diagram of BF ILC with continuously calculated Ψ

The dimensions of the matrices are as follows:

- Q, dimension $m \times m$
- L, dimension $m \times N$
- Θ, dimension $m \times 1$
- Ψ dimension $N \times m$

1.4.1 Calculating Q and L

The following applies to both methods: W_e, W_f and W_{df} are user-defined weighting matrices with dimensions $N \times N$ with the following meaning [14]

- W_e performance
- W_f robustness to model uncertainty
- W_{df} convergence speed and sensitivity to trial varying disturbances

The choice of weighting matrices affects performance, robustness and convergence of learning.
The values of weighting matrices are usually chosen as a numerical multiple of the Identity matrix.

Example values are: [8]

$$W_e = 10^6 \cdot I \ , \ W_f = 0 \ , \ W_{df} = 10^{-3} \cdot I$$

A common method

The matrix J with dimensions $N \times N$ is calculated as the impulse response matrix of the process sensitivity SG. This matrix is a lower triangular matrix for causal systems. [14]
The matrices L and G are calculated as :

$$L = (\Psi^T (J^T W_e J + W_f + W_{df})\Psi)^{-1} \Psi^T J^T W_e$$

$$Q = (\Psi^T (J^T W_e J + W_f + W_{df})\Psi)^{-1} \Psi^T (J^T W_e J + W_{df})\Psi$$

Alternative method

In some cases, the common method can lead to very long convergence due to system dynamics. [8] In these cases, an alternative form can be implemented, which should lead to a shorter convergence time. J is a matrix with dimensions $N \times m$. The k th column of the matrix J corresponds to the response of the GS transfer to the k th basis function.

$$L = (J^T W_e J + \Psi^T (W_f + W_{df})\Psi)^{-1} J^T W_e$$

$$Q = (J^T W_e J + \Psi^T (W_f + W_{df})\Psi)^{-1} (J^T W_e J + \Psi^T W_{df} \Psi)$$

2. Simulations in Matlab

2.1 Model used in simulations

2.1.1 Creating model of controlled system

My goal was to create a model of an electric motor. To create the model, I used the principle of cascade control by considering an inner loop that represents the model of an actuator - a transfer from the desired to the real current.

The controlled system is represented by transfer function 1/(T s+1). Input of the system is voltage generated by the amplifier and the output is current. The time constant representing medium sized modern servo drives is 0.3 ms.
The PI regulator was designed to represent the electronic part of the system- transfer from desired to real voltage. The design requirements were:

- Robust Stability: Gain margin > 2, sm > 0.5, Phase margin > 45 deg
- Bandwidth of closed loop 1kHz (6200 rad/s)
- Both the system and the regulator were considered to be continuous.

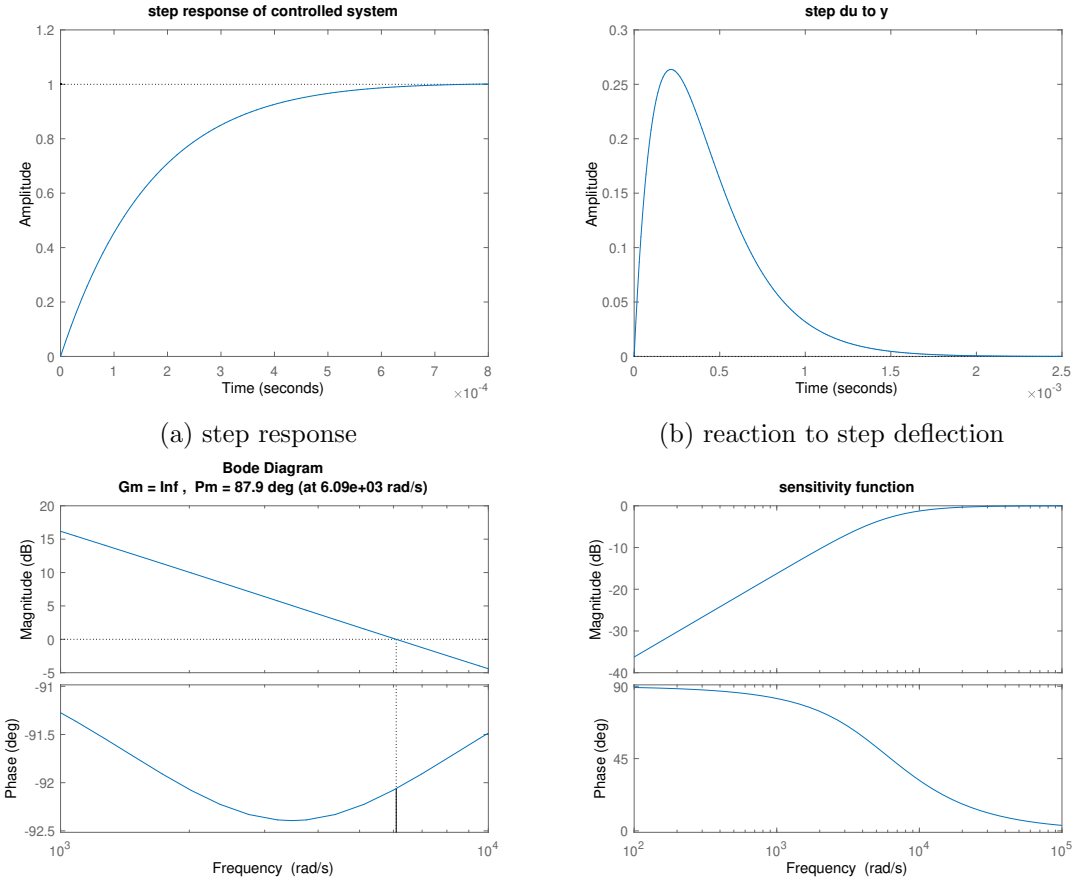
The pole placement method was used to design the PI regulator. The transfer function of the system is:

$$F_s = \frac{1}{0.0003s + 1}$$

The regulator was implemented as

$$F_r = \frac{b_1 p + b_0}{p}$$

The poles of the closed loop were chosen as $p_1 = -4650, p_2 = -4650$ in order to meet the requirements. Constants b_0, b_1 of the regulator were calculated by comparing chosen poles with poles of the regulated system as $b_0 = 6486.75, b_1 = 1.79$. I plotted multiple graphs to prove that the requirements were met.



(a) step response

(b) reaction to step deflection

(c) margin

(d) sensitivity function

Figure 2.1: System regulated by PI regulator

The required robust stability and bandwidth was proven in Figure 2.1. This controlled system represents a transfer from the required current/momentum to the real output of current/momentum.

Transfer from the required momentum to the speed of the connected weight is represented by the first order dynamical system (ideal rigid system without resonance) with the transfer function

$$P_m = \frac{1}{T_m s + 1}$$

The constant T_m has a typical value of 0.05. This transfer function represents the dynamic of the actuator. The quantity we want to control is position. The transfer from speed to position is done by the integrator. The complete transfer function of mechanical part is

$$P = \frac{Pm}{s}$$

The last part of the system is the filter, which limits gain on higher frequencies. The transfer function of the filter is

$$P_f = \frac{1}{T_f s + 1}$$

The constant T_f has a typical value of 0.001.

The overall dynamic of the system consist of the dynamic of filter, the dynamic of actuator and the dynamic of mechanical part.

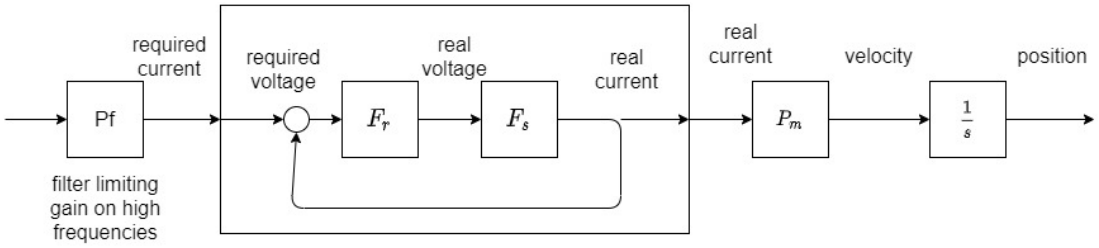


Figure 2.2: Model of system

The dynamic can be represented as 5th order transfer function:

$$P_c = P_f Ti \frac{P_m}{s} = \frac{1.2 \cdot 10^8 (s + 3624)}{s(s + 1000)(s + 4650)^2 (s + 20)}$$

2.1.2 Designing PID regulator

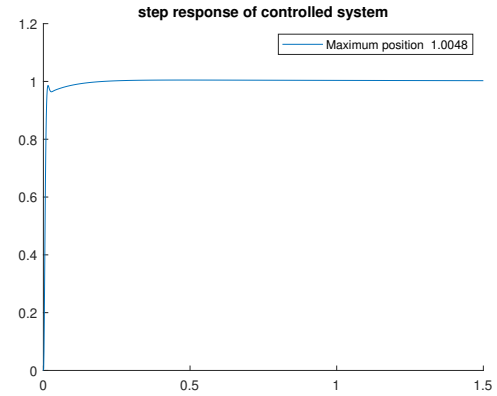
ILC is usually combined with a PID regulator. A discrete parallel PID controller with formula

$$P + I \cdot T_s \frac{1}{z - 1} + D \frac{N}{1 + N \cdot T_s \frac{1}{z-1}}$$

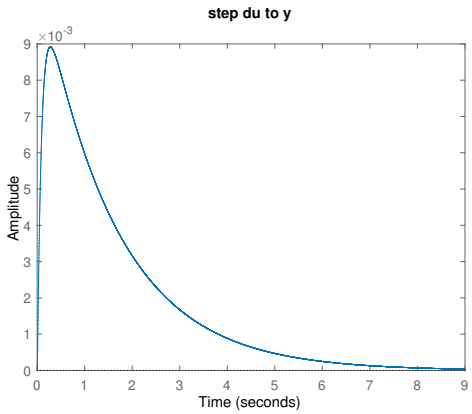
was designed. T_s - period of regulation is 0.001 s. My goal was a PID controller without overshoot in step response and the widest possible bandwidth. The designed controller also meets the standard requirements for robustness (Gain margin > 2, sm > 0.5, Phase margin > 45 deg)

My final design has 0.5% overshoot and a bandwidth of 280 rad/s. 0.5% overshoot should not cause any problems. The parameters of my design are

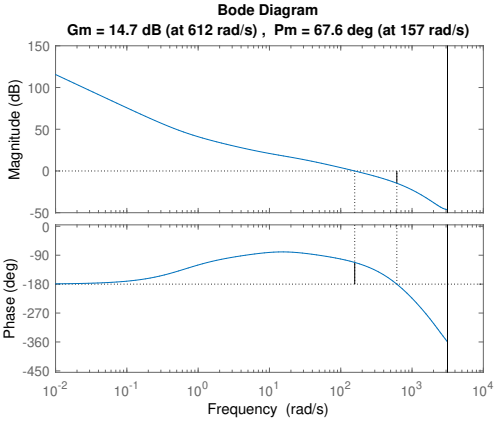
$$P = 100 \quad I = 60 \quad D = 8 \quad N = 600$$



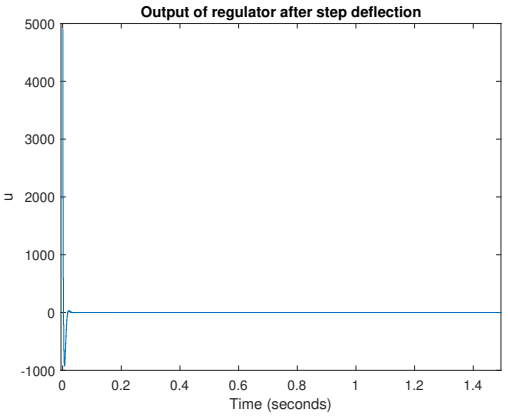
(a) step response



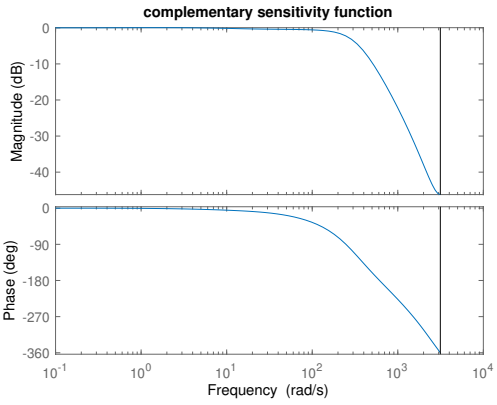
(b) reaction to step deflection



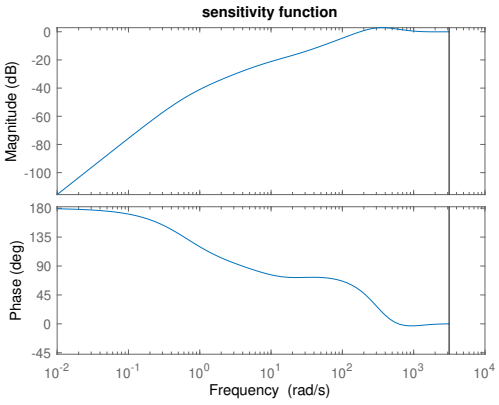
(c) margin



(d) output of regulator after step deflection



(e) complementary sensitivity function



(f) sensitivity function

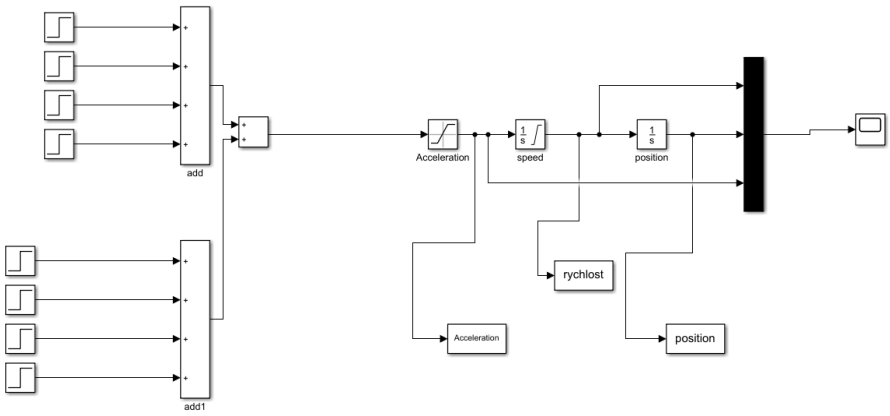
Figure 2.3: System regulated by PID regulator

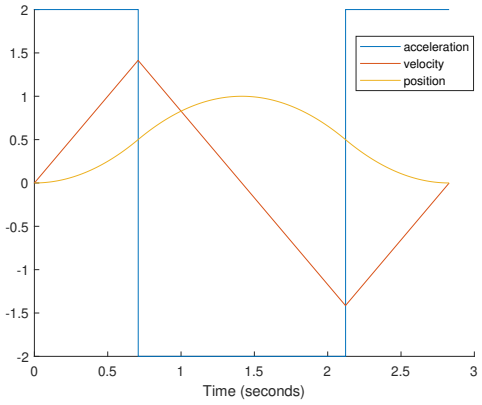
2.2 Generating reference signal

The next step is generating the trajectory of the system. I created a function, which calculates the trajectory based on the required rest position, maximum speed and acceleration. The system will follow the trajectory from a zero position to a defined final position and back. There is an unlimited number of trajectories between these two positions. My trajectory should be easy to generate and must meet the following requirements: The system starts with zero speed and should end with zero speed, both in defined position and at the end. The speed must be continuous.

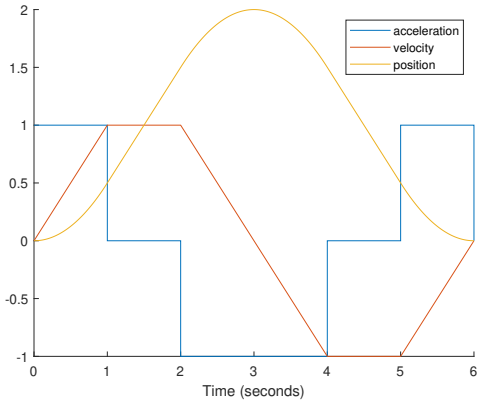
My function calculates the parameters of the simulation and then runs a Simulink simulation. I used 2 integrators to generate this trajectory. The first integrator integrates acceleration to get speed and the second integrates speed to get position. The input of the first integrator switches between positive and negative maximum acceleration and zero. The trajectory starts with constant acceleration motion. Then it either continues with constant deceleration motion after reaching half of the designated position (Fig 2.5.a), if the maximum speed is high enough, or switch to constant motion and then constant deceleration motion (Fig 2.5.b) if the maximum speed is reached.

Figure 2.4: Simulation scheme





(a) $p_{end} = 1$ $v_{max} = 2$ $a_{max} = 2$



(b) $p_{end} = 2$ $v_{max} = 1$ $a_{max} = 1$

Figure 2.5: Calculated trajectories

2.3 Feedforward control

Because we have not only the position, the system should follow, as well as speed and acceleration of the system. Therefore feedforward control based on speed and acceleration can be implemented. Feedforward control was implemented as linear combination of the speed and acceleration. The signals from the speed and acceleration will go through the gain, which allows us to manually tune gain constants k_a, k_v to optimise system error.

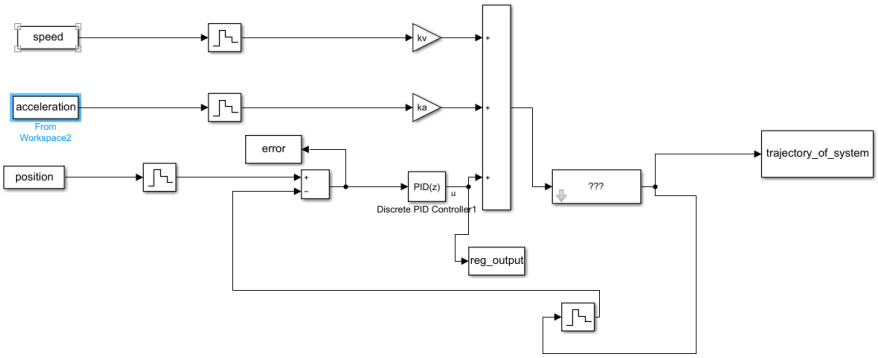


Figure 2.6: Simulation scheme

2.3.1 Finding optimal values of constants

The lowest RMSE was achieved with $kv = 0.9971ka = 0.0512$. The feedforward control with these constants acts as inverse to the slowest mechanical part of the system. The ideal inverse would be

$$P_{inv} = \frac{T_m s^2 + s}{1}$$

With this we get rid of the dynamic of the mechanical part of the system, which is much slower than other parts.

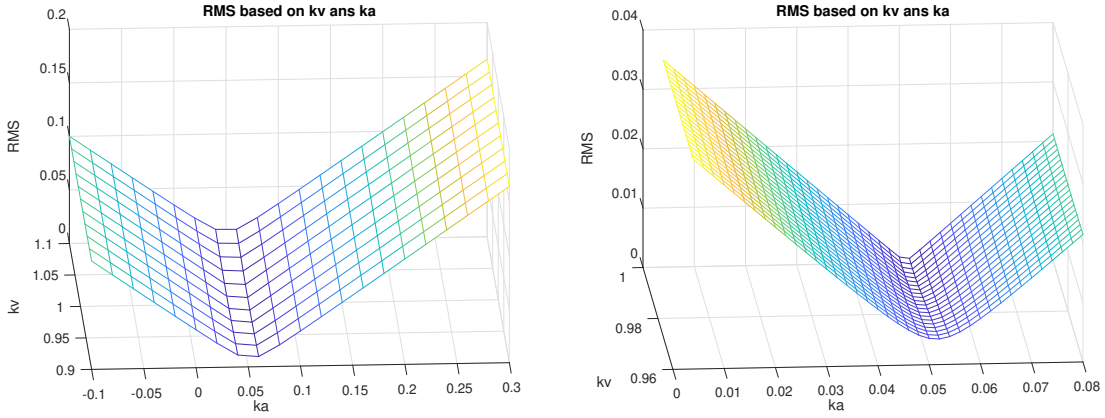


Figure 2.7: RMS based on k_v and k_a

2.3.2 Comparing combined and feedback - only control

With feedforward control the RMSE of the system regulated with a combination of feedforward regulation and feedback PID regulation was more than 100 times lower than the RMSE of the system regulated only by the PID regulator.

	Trajectory 1	Trajectory 2
	RMS	RMS
without feedforward control	0.0171	0.1786
with feedforward control	0.000047495	0.00037356

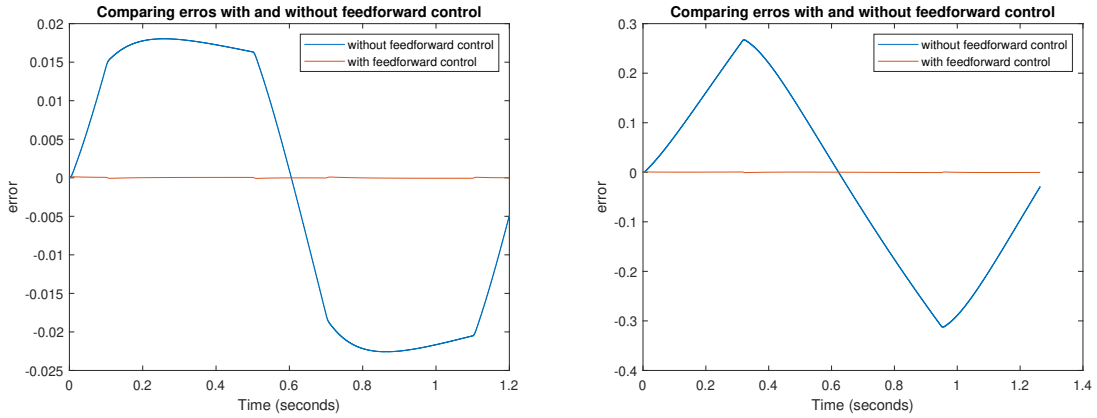


Figure 2.8: Comparison of combined and feedback control

2.4 Simulating ILC in Matlab

The ILC simulation consists of a Matlab script, which calculates the ILC vector and computes and records the simulation parameters, as well as a model in Simulink. The script first initializes the variables and calculates the desired trajectory, then repeatedly calls the simulation in Simulink, evaluates its result and recalculates the ILC vector. A Simulink model with the following scheme was used to simulate the ILC algorithms in Matlab.

The model uses a PID controller computed in 2.1.2 and a controlled system computed in 2.1.1 .

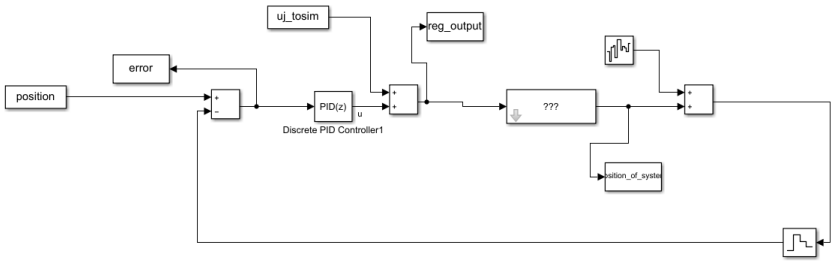


Figure 2.9: Simulation scheme ILC

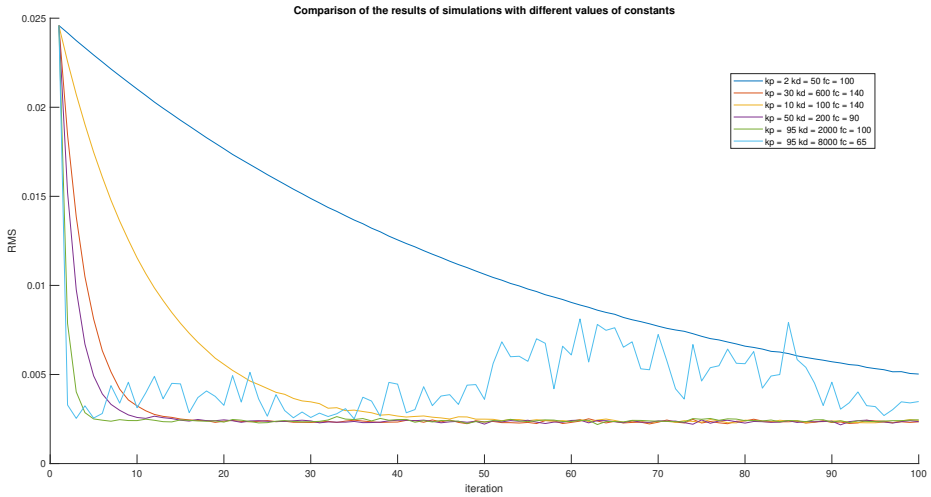
In the model, noise is added to the system output in order to simulate the noise of the real system. Uj_tosim represents the ILC vector calculated in the script.

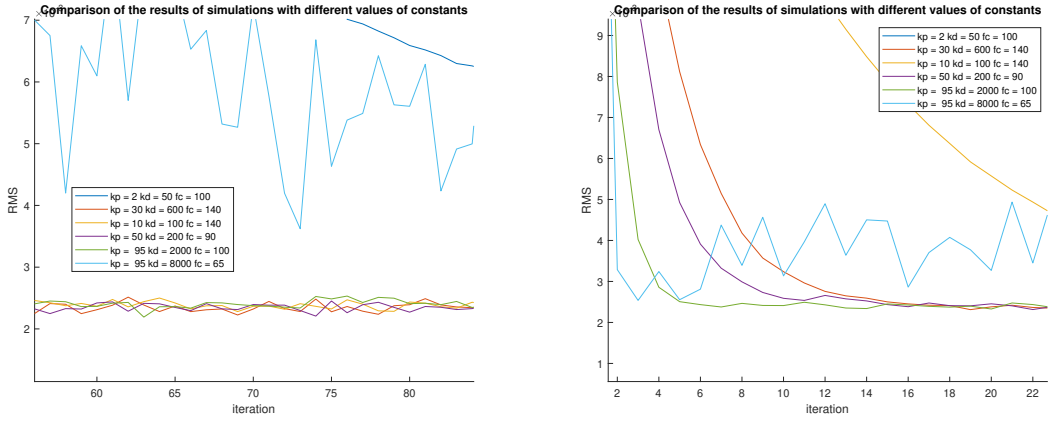
2.4.1 Simulating PD-ILC

When implementing PD-ILC I proceeded as follows: Different combinations of the value of learning constants k_p and k_d were tested using manual tuning. From these combinations, some combinations were selected to represent different types of behaviour. For these selected combinations, a second-order low- pass Butterworth filter was designed with a maximum bandwidth that guaranteed theoretical convergence if convergence could be achieved. Zero-phase digital filtering in the form of Matlab's filtfilt function was used for the Q filtering.

Comparison of the results of simulations with different values of constants shows the following: Higher values of k_p and k_d lead to faster convergence, but may also cause higher

asymptotic error, probably due to more noise amplification, which may lead to undesired behavior. Due to the unknown value of noise in a real system, it would probably be better to choose, at least in the first implementation, lower values of constants.
In the following plots and table, fc means the cutoff frequency of the Q filter.





(a) zoom1 (b) zoom2

Figure 2.10: Comparing different values of k_p, k_d and bandwidth

Theoretical convergence of the PD-ILC algorithm				
kp	kd	fc	convergence with Q	convergence with Q^2
2	50	100	Yes	Yes
10	100	140	Yes	Yes
50	200	90	Yes	No
30	600	140	Yes	Yes
95	2000	100	Yes	Yes
95	8000	65	Yes	Yes

For some combinations of parameters k_p and k_d I could not find a value of the cutoff frequency fc that would imply theoretical convergence.
Of the simulated parameter combinations, $k_p = 30$ $k_d = 600$ $fc = 140$ looks the best.

It has a low asymptotic error and a relatively high convergence rate. It also satisfies the theoretical convergence conditions.

2.4.2 Simulating FD-ILC

For FD ILC implementation, the L filter was first calculated using the ZPTEC method. Then, a second-order low-pass Butterworth filter with a suitable cutoff frequency was designed to ensure convergence. The resulting filter L was converted to the form time shift multiplied by L_c and the L filtering was implemented as a filtering of the shifted vector by the L_c filter - see 1.2.1 . The Matlab filter function was used for L filtering. Zero-phase digital filtering in the form of Matlab's filtfilt function was used for the Q filtering. The first and last few samples of the resulting control vector were changed to zero to avoid undesired behavior.
To demonstrate the correct implementation of the ZPTEC algorithm in the calculation of the L filter, the Bode plot of GSL transfer function was plotted.

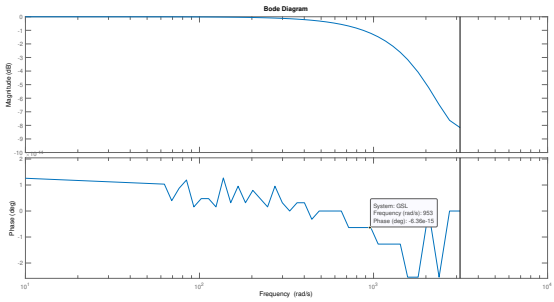
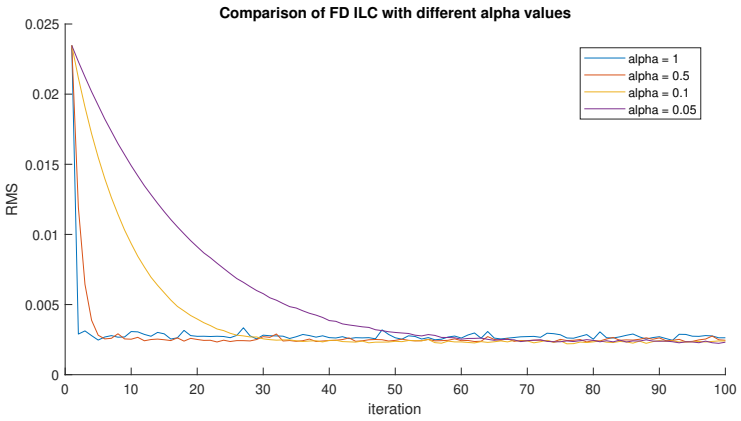
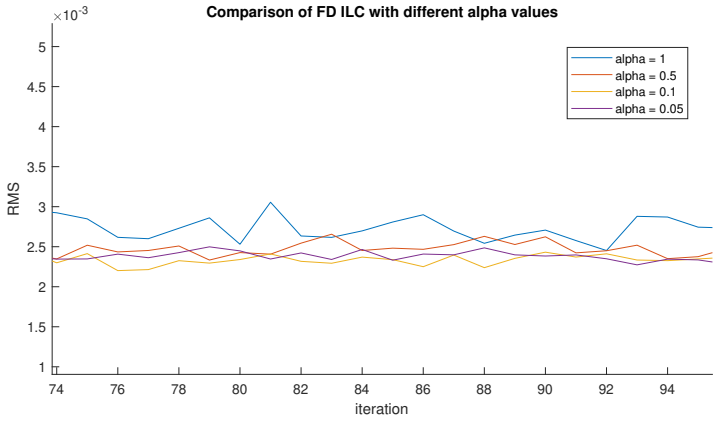


Figure 2.11: Bode plot of GSL

The RMS values during the iterations were compared for different values of the learning constant α. It can be seen in the plot that higher values of α lead to faster convergence, but also to a higher asymptotic error , due to higher noise amplification. $\alpha = 1$ leads to convergence after one step, but also to a relatively high asymptotic error. $\alpha = 0.1$ is probably the best choice of the tested options due to similar asymptotic error as $\alpha = 0.05$. but significantly faster convergence.



(a) RMS



(b) zoom

Figure 2.12: Comparing RMS with different α

2.4.3 Simulating BF-ILC

I chose $\dot{r}$ and $\ddot{r}$ as basis functions - velocity and acceleration. I first tried to implement BF-ILC using the common method of computing the matrices Q and L. This implementation had a disproportionately long convergence time. I then implemented BF-ILC using the alternative method of computing the matrices Q and L. This implementation had a good convergence rate and therefore I used it for all the simulations in Chapters 2 and 3.

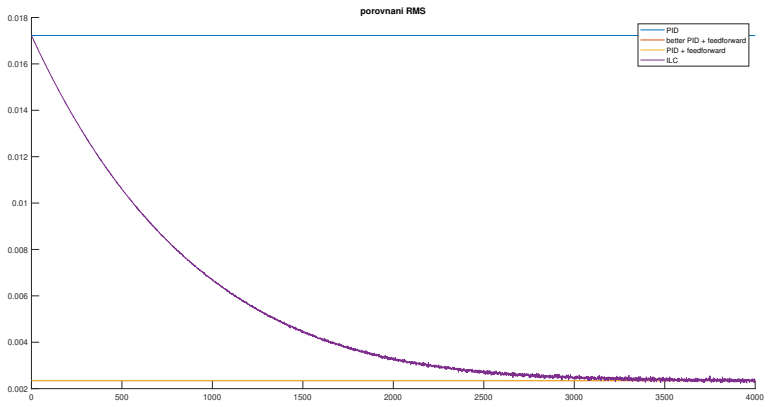
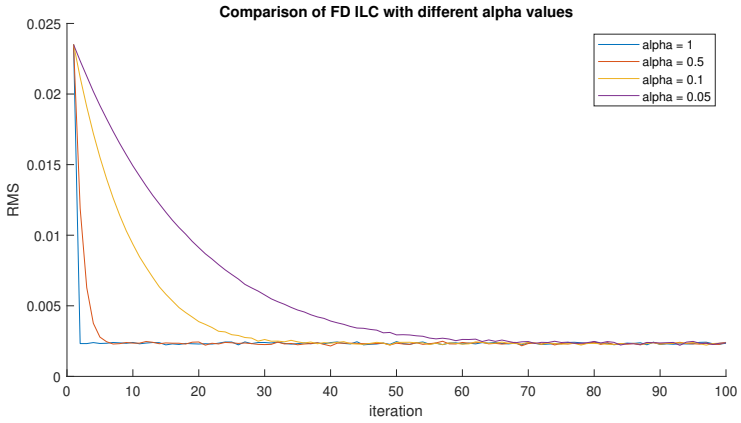
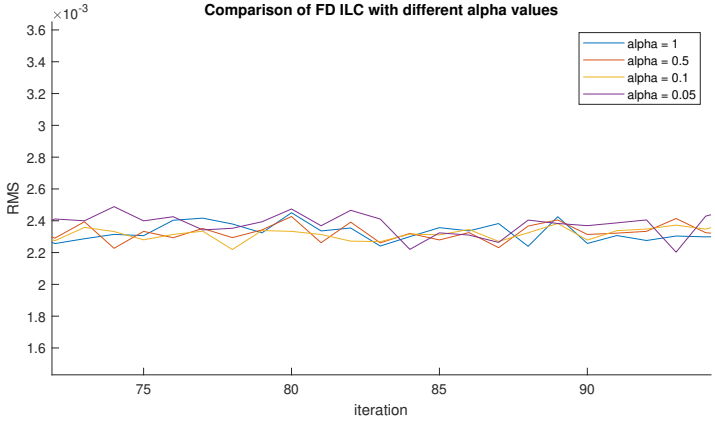


Figure 2.13: BF ILC with matrices calculated by the common method

Comparing the BF ILC with different values of the learning constant α, we see that a higher value of alpha leads to faster convergence but may lead to a larger response to noise. The asymptotic errors correspond to the error when using feedforward control with ideal values of constants.



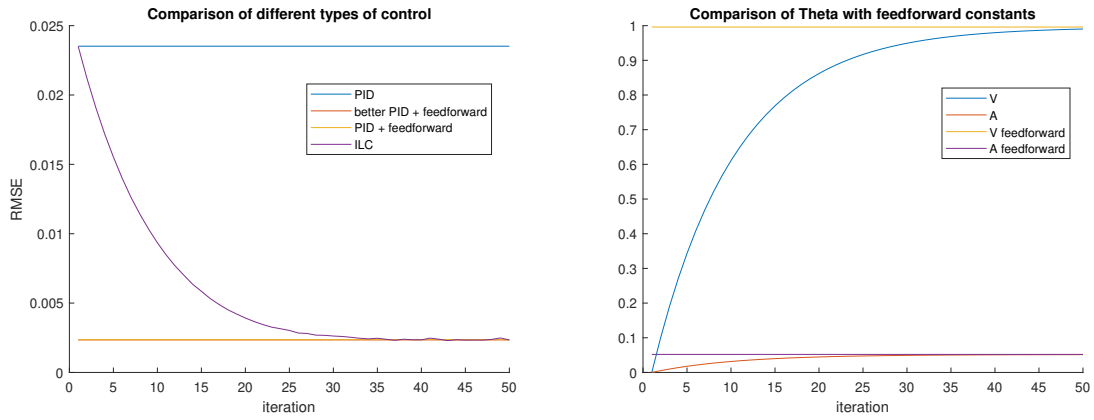
(a) RMS



(b) zoom

Figure 2.14: Comparing RMS with different α

Since feedforward control was implemented from velocity and acceleration - my chosen basis functions, the value of the constants k_a, k_v calculated in Section 2.3 can be compared with the values of the coefficients Θ. It can be seen that the values of the corresponding constants Θ converge to the values of the 'ideal' constants calculated in 2.3 .



(a) RMS (b) Theta

Figure 2.15: Results of simulation with $\alpha = 0.1$

2.4.4 Comparison of different ILC

I compared the performance of PD, FD and BF ILC with the following parameters:

- PD ILC: $k_p = 30$ $k_d = 600$ $fc = 140$
- FD ILC: $\alpha = 0.1$
- BF ILC: $\alpha = 0.1$, Alternative form

It can be seen in the plots that for the same $\alpha = 0.1$, BF and FD ILC converge in a similar way. PD ILC has a slightly higher convergence rate than FD and BF ILC, mainly due to the choice of learning constant $\alpha = 0.1$. The asymptotic error of all variants is approximately the same.

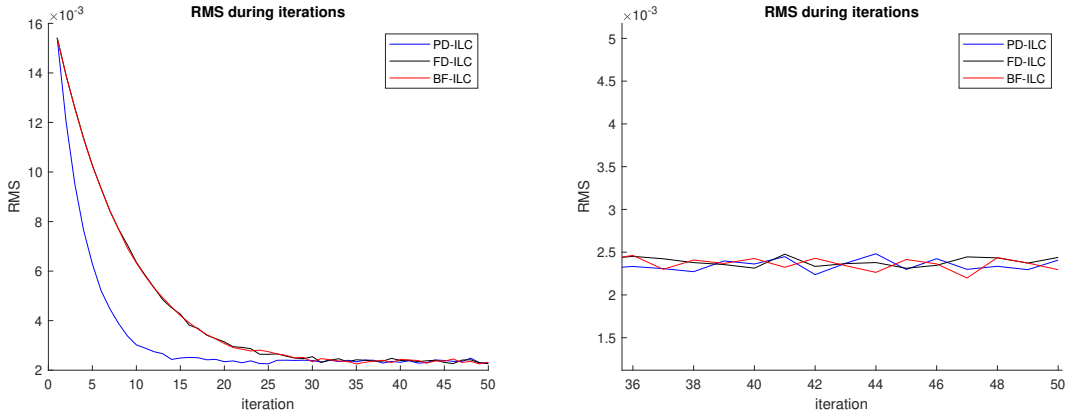


Figure 2.16: Comparison of different ILC variants, stable reference trajectories

If we change the reference trajectory, the FD and PD ILC have to start from the beginning - the error values increase significantly. The BF ILC retains the learned Θ values, so there is no significant change in error values. However, the L and Q matrices must be recalculated.

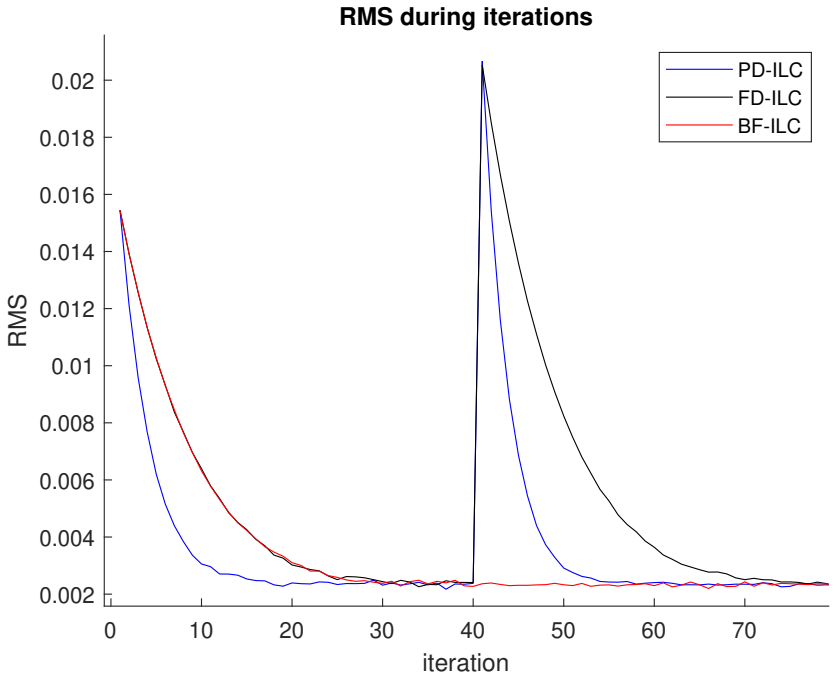


Figure 2.17: Comparison of different ILC variants when the reference trajectory changes

3. Implementation in Rexygen

3.1 Introduction of Rexygen

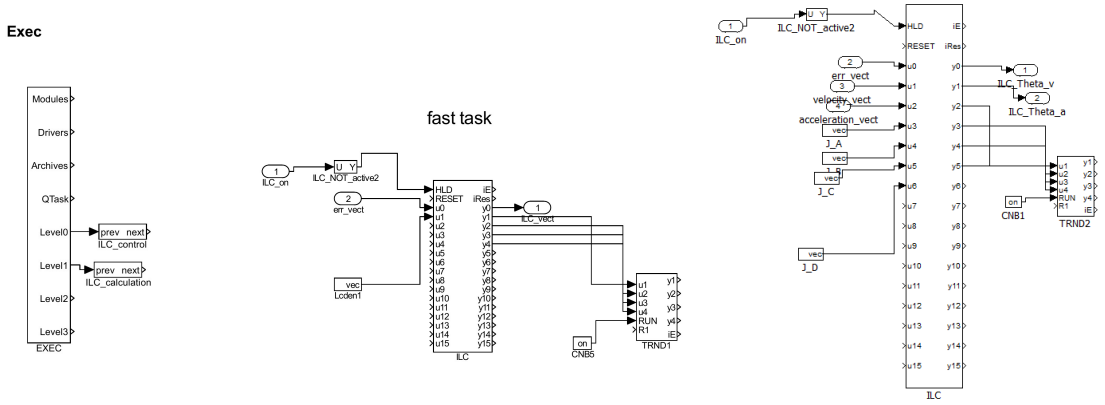
REXYGEN is an advanced tool for process control, robotics and diagnostics. The RexCore runtime core runs on top of the operating system, coordinating the execution of algorithms and providing access to input and output signals. REXYGEN supports many different platforms and is platform independent. Rexygen's main development tool is REXYGEN Studio, which acts as a unified development environment for all platforms. Programming in Rexygen is done visually using function blocks that can be selected from an extensive library of function blocks. [3] Rexygen is developed by REX Controls, which has a long-term cooperation with the NTIS research centre of the Faculty of Applied Sciences of the University of West Bohemia in Pilsen.

3.2 Implementation of ILC in Rexygen

3.2.1 Scheme of implementation

The implementation in REXYGEN consists of three files - an exec file, a fast-task and a slow-task. The fast-task runs every 1ms and contains all the functionality that needs to be executed in real time, while the slow-task is run after 100ms and contains the ILC calculation.

The exec file contains the EXEC block and acts as the main project file. The EXEC block is the cornerstone of the REXYGEN system. When compiled, the EXEC block and the blocks attached to it determine what and how will be part of the compiled program. In our case, there are two TASK blocks attached to the EXEC block. These TASK blocks represent our tasks using the filename parameter, which acts as a reference to the corresponding source file. The EXEC block also defines the tick length (how often each task will be executed) for each level.



(a) Exec file (b) slow-task PD ILC (c) slow-task BF ILC

Figure 3.1: Implementation in REXYGEN

The slow-task contains a REXYGEN block in which the ILC calculation takes place and blocks containing the matrices that are required to calculate the corresponding ILC.

These matrices are computed in Matlab and then stored in the CNA block. A Boolean indicating that the ILC is enabled is fed from the fast-task to the HOLD pin of the PYTHON block. As a result, the calculation in the slow-task does not proceed if the ILC is not enabled. Furthermore, the error vector is fed into the slow-task from the fast-task. In the case of BF ILC in addition to the error vector, the velocity and acceleration vectors are fed in. The slow-task also contains a TRND block which is used to store the data computed in the PYTHON block- RMSE, the number of iterations, and highest and lowest measured error.

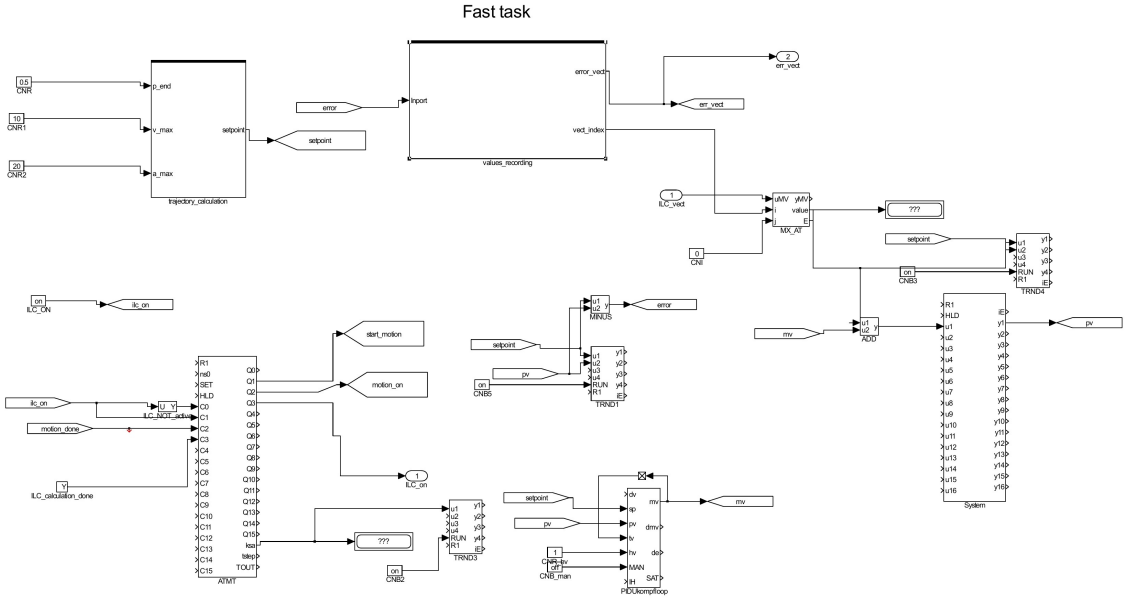


Figure 3.2: Implementation in REXYGEN fast-task PD and FD ILC

The fast-task contains two subsystems. The first is responsible for calculating the desired trajectory using REXYGEN blocks:

- RM_Axis, which serves as a cornerstone for generating motion in REXYGEN. It stores and provides motion information along a single axis.
- MC_Power, which must be implemented and acts as an on/off switch.
- Two MC_MoveAbsolute blocks. One implements movement to the desired position and the other back to zero. During these movements the maximum speed and acceleration are limited to the desired values.

The second subsystem is responsible for storing the current error in a matrix, which it then passes by reference to the slow-task. The first output of the subsystem is a reference to the error vector. The second output of that subsystem is the number of ticks since the motion started in the current iteration. Tick counting is performed by the integrator, which resets at the start of the motion and stops at the end of the motion. The current error value is stored in the MX_MAT block using MX_DSASET. The position at which the error is to be stored is determined by the integrator output. MX_DSASET also provides a reference to the stored data, which is passed to the first subsystem output. For the BF ILC, the velocity and acceleration information is stored in the same way.

In the fast-task there is also a CSSM block, which represents and simulates the real system; a PIDU block, which contains the PID controller calculated in 2.1.2; the TRND blocks which are used to write trends; the ATMT block, which is responsible for the

runtime control program, as will be explained later and the MX_AT block. MX_AT allows us to extract individual elements from the ILC vector that was computed in the fast-task. The current tick information from the second subsystem is used to determine the correct element that needs to be fed into the system. In BF ILC, instead of the ILC vector, the calculated values of the constants Θ are fed into the fast-task. The ILC control is then calculated by multiplying the current velocity and acceleration by the corresponding constant.

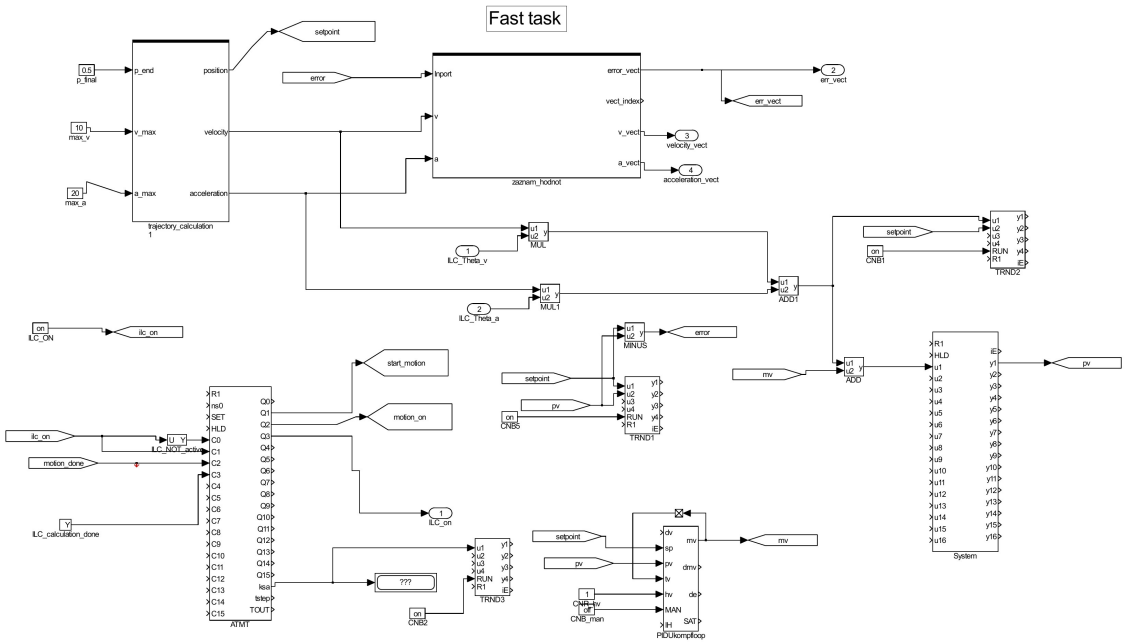


Figure 3.3: Implementation in REXYGEN fast-task BF

3.2.2 Program runtime control using ATMT

The finite state machine implemented in the ATMT block [1] is used to control the execution of the program. The automaton is defined by a table of transitions in which each line represents a transition rule. Line

$$S_i \;\; C_j \;\; FS_k$$

means if the current state is S_i and at the same time the transition condition C_j is satisfied it goes to the state S_k. The values of the transit conditions are determined by the binary input of the ATMT block. Output Q_n of the block is set to True if the current state of the block is n. The states and the transitions between them used to control program execution are represented in Figure 3.1. [12]

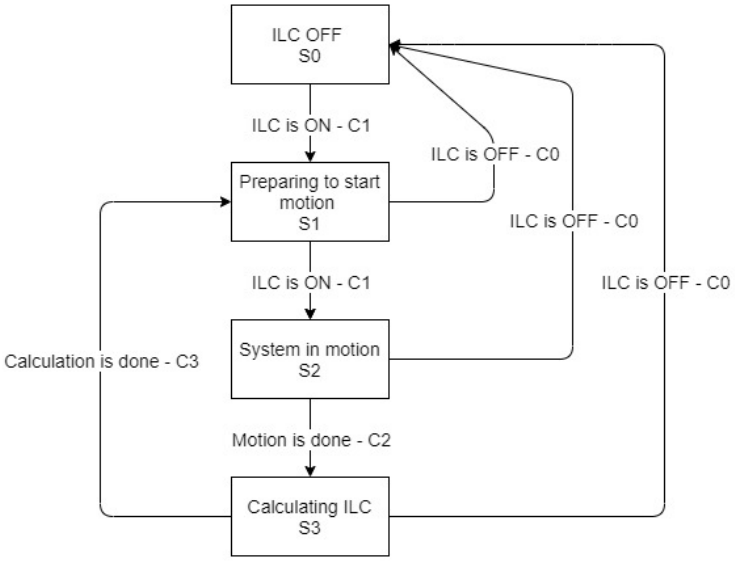


Figure 3.4: States and transition of ATMT

3.2.3 Python block

A PYTHON block [2] allows the user to implement a user-defined function. The PYTHON block performs functionality written in a Python script, which name is specified as a block parameter. Compared to the other way of writing user-defined functions in REXYGEN — the REXLANG block — the PYTHON block allows the user to develop more easily by programming in a high-level programming language. The PYTHON block also supports third-party libraries available in Python. The disadvantage of the PYTHON block can be instability in some corner cases. There must be four functions in a Python script:

- **main()** is executed whenever the block is executed
- **init()** is executed at startup and after resetting the block
- **parchange()** is executed whenever a parameter of the block is changed
- **exit()** is executed when the control algorithm is stopped

Data exchange between the script and the REXYGEN system is done through a PyRexExt module . This module includes a REX object, which is used for data exchange operations. The reading value of input or parameter and the writing value of output is done by **.v** property. When using the **.v** property, there is an automatic conversion between REXYGEN and Python data types. The REX object can be used to create a handle to an external REXYGEN item and to write messages to the system log.
The data types of output signals, input signals and parameters must be specified in the configuration file. If the configuration file is not found when the block is started, a new one is created with the default data type double for all inputs, outputs and parameters. The configuration file can be edited using the tool available in the PYTHON block. In addition to the common data types(int, float, double, string, array), the inputs and outputs also support NumPy array. The NumPy array on the output is automatically converted to a regular array. Regular array on input is automatically converted to NumPy array.

3.3 Implementing ILC with PYTHON block

As mentioned earlier, due to the limited computational capabilities of Python, I calculated some of the things needed to calculate the ILC in Matlab. The Q filter coefficients for the

PD and FD ILC and the L filter coefficients for the FD ILC were calculated. For the BF ILC, the matrices representing the GS transfer as a state space model were calculated in order to compute the Q and L matrices. PD ILC and FD ILC were implemented according to the functions described in the first chapter in a similar way as in Matlab. The BF ILC has been implemented with an alternative method of computing the matrices Q and L. Unlike the Matlab implementation, the feedforward control is not computed in a Rexygen block, but is instead generated in a fast-task based on the computed parameters Θ .

3.3.1 IIR filter

The IIR filter used in the FD ILC was implemented in transposed direct form 2. The implementation was verified by comparing the filtering results using the filter function in Matlab against my implementation. [15]

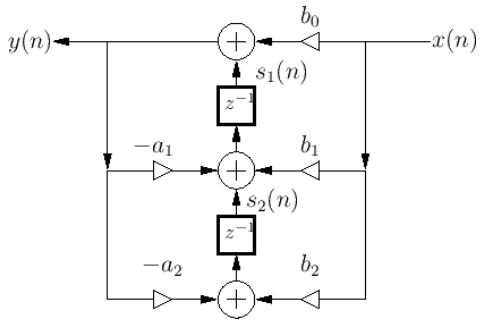


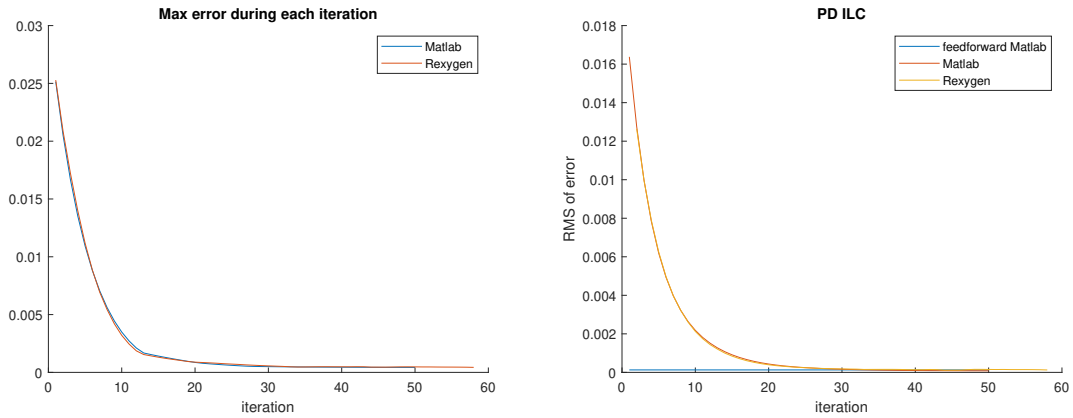
Figure 3.5: Transposed direct form 2 implementation of a second order filter [15]

3.3.2 FIR filter

Due to the lack of a filtfilt algorithm in Python, I used a zero-phase finite impulse response filter for the Q filtering.

3.4 Comparing implementation in Rexygen with Maltab

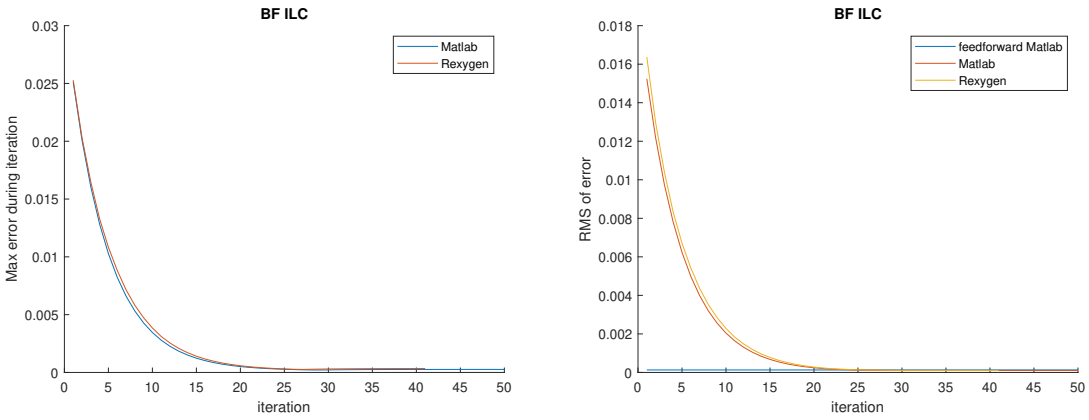
I compared the simulation results in Rexygen with the simulation results in Matlab and Simulink to verify the correctness of the Rexygen implementation. The simulations were performed with the same constants and FIR filters. The calculated errors are very similar. so the implementation can be considered correct.



(a) Max error

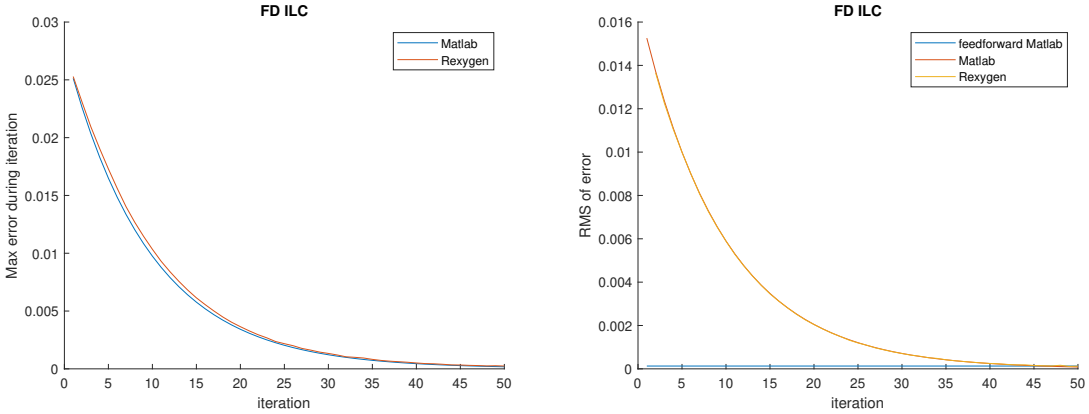
(b) RMS

Figure 3.6: Comparison of results in Matlab and Rexygen PD ILC



(a) Max error (b) RMS

Figure 3.7: Comparison of results in Matlab and Rexygen BF ILC



(a) Max error (b) RMS

Figure 3.8: Comparison of results in Matlab and Rexygen FD ILC

4. Controlling real mechatronic system

4.1 Description of the controlled system

The motion system consists of an electrical drive (590 W permanent magnets synchronous motor driven by a servo amplifier), flexible coupling, bearing housing, inertia flywheel. [11] The system has one degree of freedom - it rotates around a vertical axis. The system is controlled by an industrial PC running hard real-time Linux-based software environment with a REXYGEN control system. The system is connected to the computer via EtherCAT communication with a 5 - kHz update rate.

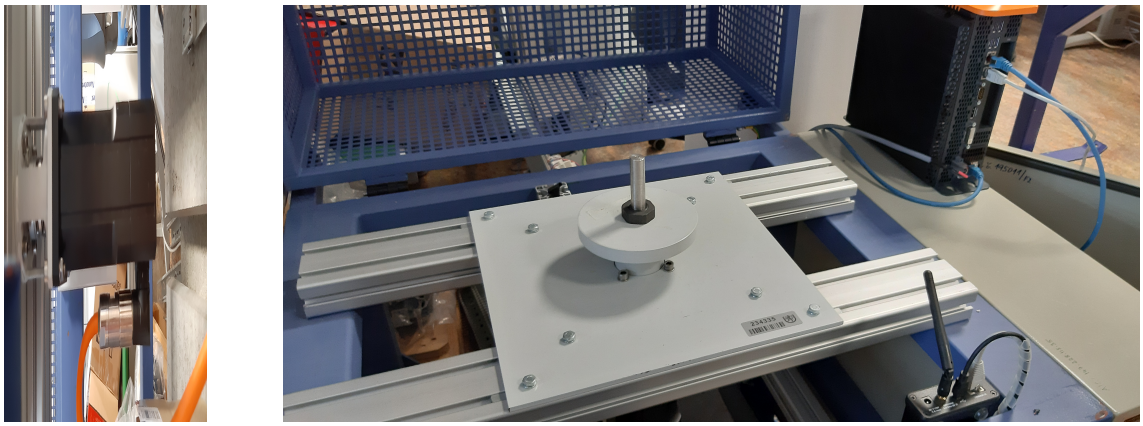


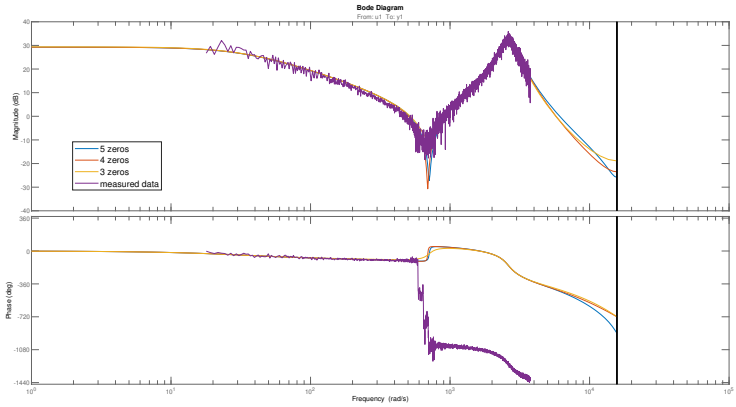
Figure 4.1: Controlled system

4.2 System identification using measured data

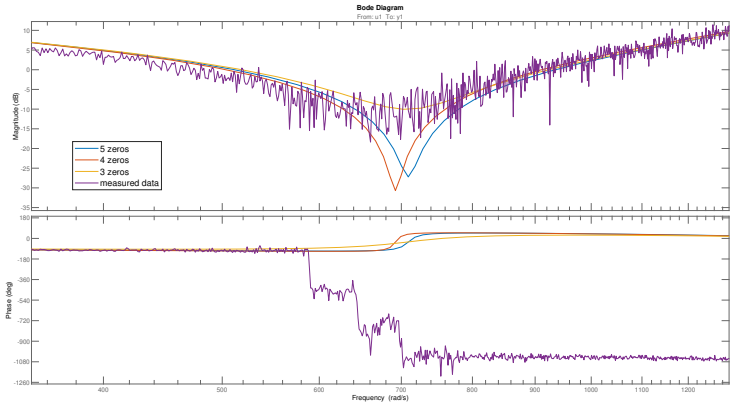
Data representing the dynamics of the transfer from the moment of motor to speed were measured for different frequencies. I also have information that the computer that controls the system can work with a sampling period of 0.2ms - a sampling rate of 5kHz. There is also a known delay due to the data transfer between the computer and the drive amplifier. This delay is three samples, so I designed the transfer function with a constant input output delay of 3 samples during the identification. I made the identification using the System Identification Toolbox Matlab. I designed discrete transfer functions with a sampling period of 0.2ms and a constant input/output delay of 3 samples and varying number of zeros and poles. From these experiments I found that the appropriate number of poles is 5. 5 is the smallest number of poles that gives an accurate representation of the measured data. I got a very similar identification error for 3,4 and 5 zeros. When comparing the resulting transfer functions with the measured data, the 3 zeros option looks best. At the same time, the lower complexity of the model should facilitate the control design. I multiplied the identified system by an integrator $\frac{T_s}{z-1}$ to get the transfer from motor moment to position. The resulting transfer function is:

$$z^{-3}\frac{0.0001308z^4 - 0.0002554z^3 + 0.0001272z^2}{z^6 - 5.107z^5 + 11.24z^4 - 13.67z^3 + 9.691z^2 - 3.807z + 0.6503}$$

The PID controller and ILC was designed for this resulting transmission.



(a) Bode plot



(b) zoom

Figure 4.2: Comparison of similarities of identified transfer functions with 3, 4, 5 zeros against measured data.

4.3 Design of a PID controller to control the identified system.

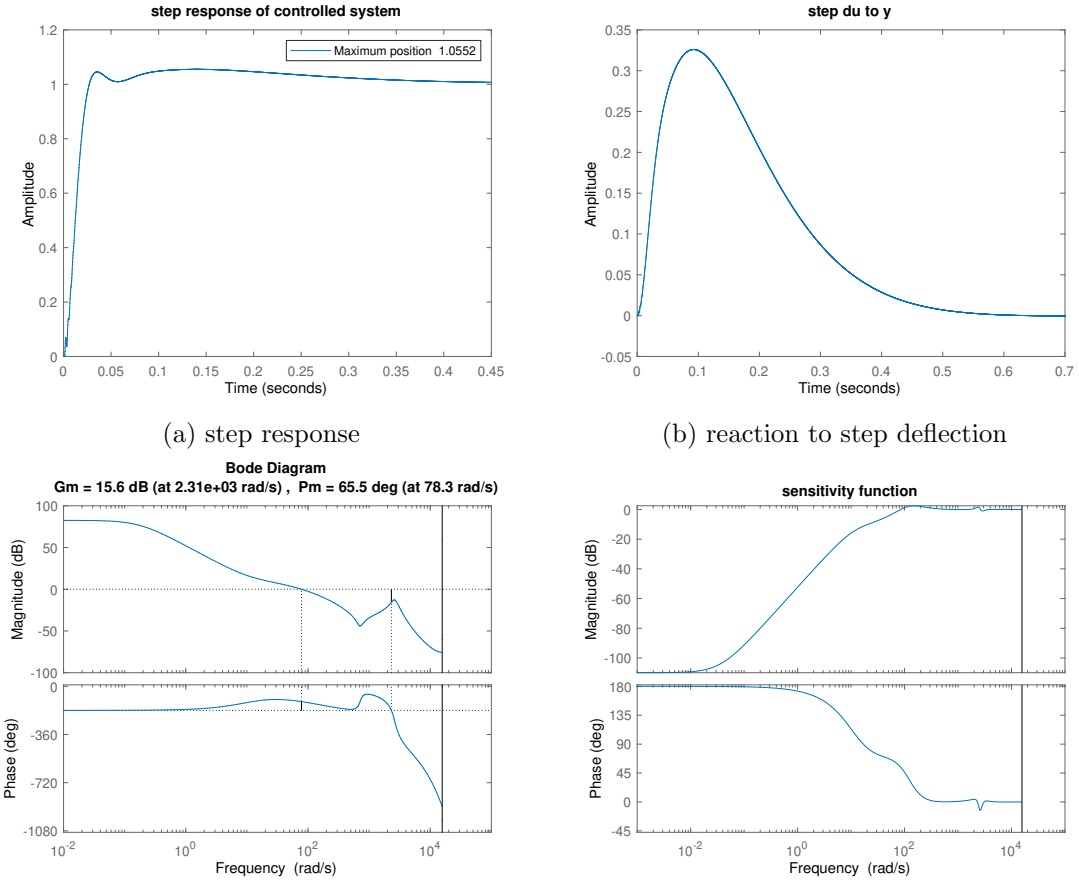
The discrete PID controller designed for the identified system has 5.5% overshoot in response to a step change in the reference trajectory. I converted the PID controller using the pidstd command into the form

$$K_p(1 + \frac{1}{T_i}\frac{T_s}{z-1} + T_d\frac{1}{\frac{T_d}{N} + \frac{T_s}{z-1}})$$

with parameters

$$K_p = 2.25,\ T_i = 0.161,\ T_d = 0.0356,\ N = 5.33,\ T_s = 0.0002$$

.



(a) step response

(b) reaction to step deflection

(c) margin

(d) sensitivity function

Figure 4.3: Identified system regulated by PID regulator

4.4 ILC implementation for real - system control

I linked the implementation of the ILC algorithms in Rexygen described in Chapter 3 to a sample application. The sample application contained the functionality needed to control the real system.

I used the identified system and the designed PID controller to determine the constants, filters, and matrices needed to implement ILC. I determined the constants k_p and k_d using manual tuning in Matlab. I obtained the L filter for FD ILC by inverting the GS transfer. The GS transfer has no unstable zeros, so there was no need to use ZPTEC.

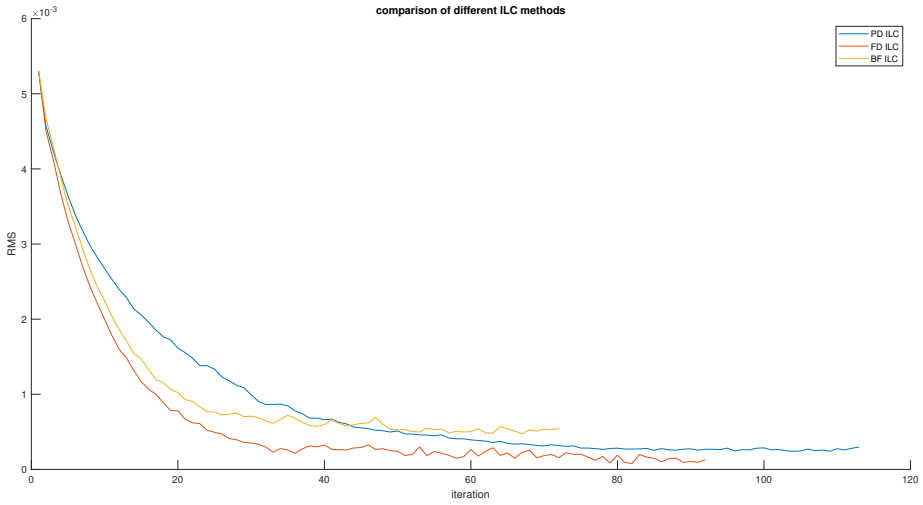
The PD ILC provided a significant improvement in control compared to pure feedback control and no change in the control algorithm was necessary.

For FD ILC, it was necessary to set the first and last few samples of the feedforward vector to zero because, due to the shift of the filtered vector, the L filter was causing undesired behavior.

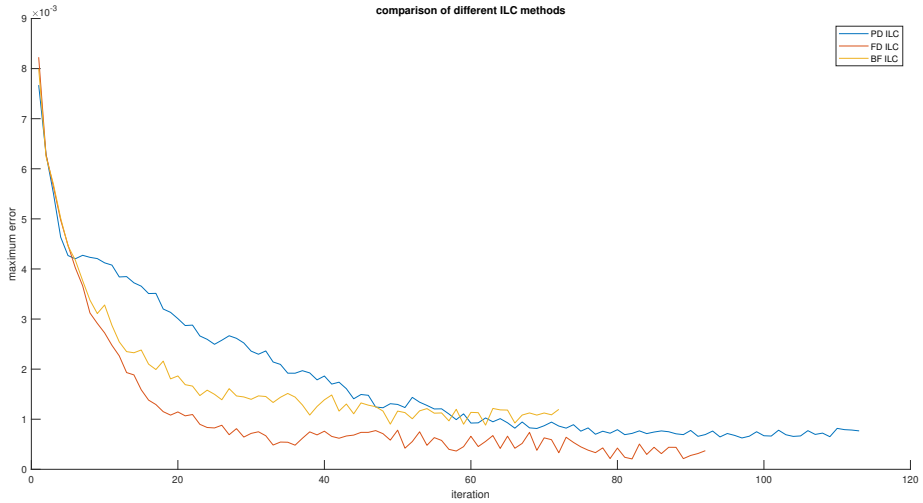
The BF ILC with two basis functions - velocity and acceleration - did not lead to sufficient improvement due to limited ability to react to the start of motion. So I added a third base function - the signum of velocity. I also chose a non-zero weighting matrix W_f to improve robustness to model uncertainty. With these changes, BF ILC had results comparable to PD and FD ILC.

4.5 Data measured on a real - system

The trajectory used for comparison had the following parameters: $p_{end} = 0.5$, $v_{max} = 4$ and $a_{max} = 10$. When comparing the responses to the trajectory change, the first trajectory had parameters : $p_{end} = 0.5$, $v_{max} = 4$ and $a_{max} = 10$. and the second trajectory had parameters $p_{end} = 1$, $v_{max} = 4$ and $a_{max} = 10$. I compared PD ILC with $k_p = 0.25$ and $k_d = 5$ with FD and BF ILC, which had $\alpha = 0.1$.



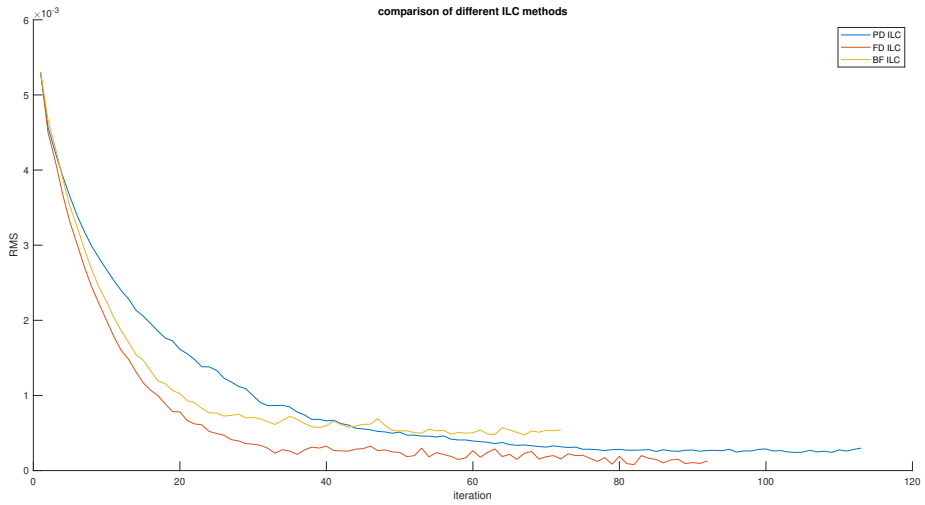
(a) RMS



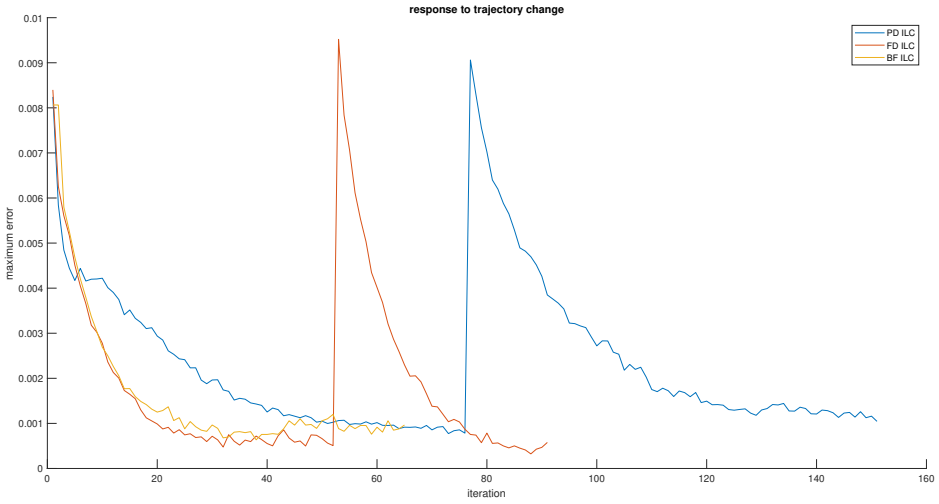
(b) Maximum error

Figure 4.4: Comparison of real-system control results using different variants of ILC

The convergence rate is similar for BF and FD ILC with the same α . This behavior is similar to the simulated behavior. The asymptotic error is slightly larger for the BF ILC.



(a) RMS



(b) Maximum error

Figure 4.5: Response to trajectory change using different variants of ILC

It can be seen that changing the trajectory causes a step in error for PD and FD ILC because the learned control vector is not appropriate for the new trajectory. The BF ILC has no problem with trajectory change.

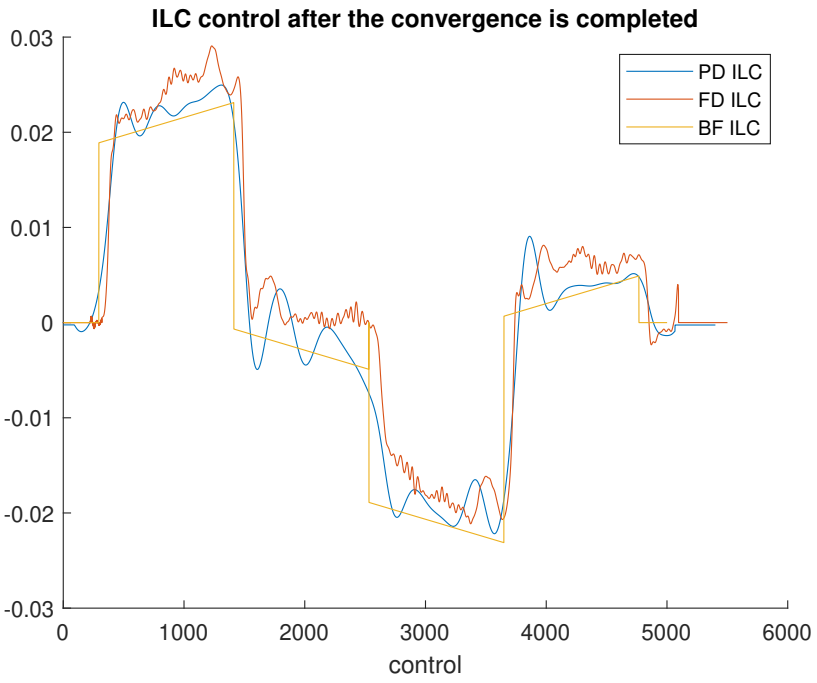


Figure 4.6: ILC control after the convergence is completed for different variants of ILC

The control generated by all forms of ILC is relatively similar.

All variants of the ILC lead to large improvements in performance. I got the smallest asymptotic error for FD ILC, probably due to a good L filter that inverts the dynamic of GS transfer well. If the system were to often change trajectory, then a BF ILC would be a good choice, as the BF ILC can adapt instantly to trajectory changes, unlike the FD and PD ILC.

5. Conclusion

In this thesis, the use of Iterative Learning Control methods to improve the performance of motion control systems was investigated. First, in the theoretical part, individual methods of Iterative Learning Control were introduced. Convergence was also analysed for some of the methods.

Simulations were then run in Matlab to test the behaviour of each method and the effect of constants on it. The effect of the constants was as follows:

- PD ILC: Higher values of k_p and k_d cause faster convergence, but may also cause unwanted noise amplification.
- FD and BF ILC Higher values of α lead to faster convergence, but can lead to higher asymptotic error.

This behavior is consistent with that described in the literature I reviewed.

Subsequently, the methods were implemented in a real-time system. This implementation was then used to control the real system. All three tested methods - PD, FD and BF ILC - improved the performance of the real system.

The main advantage of PD ILC is that it is not necessary to know the exact model of the controlled system. Manual tuning of constants, on the other hand, can be lengthy.

The main advantage of FD ILC is very good performance if the exact model of the controlled system is known. In my experiments, FD ILC achieved the lowest asymptotic error. At the same time, it is possible to get convergence in one step when choosing $\alpha = 1$. The main disadvantage is the necessity to know the accurate model of the controlled system.

A shared disadvantage of PD and FD ILC is the inability to respond to a change in trajectory - learning must start over.

The main advantage of the BF ILC is the ability to react to a change in trajectory. One disadvantage of BF ILC is its inability to generate arbitrary control - the generated control must be a linear combination of basis functions. Because of this, I had to add an additional basis function when implementing BF ILC on a real system. BF ILC also requires accurate knowledge of the model of the controlled system.

References

[1] Atmt – finite-state automaton. `https://www.rexygen.com/doc/ENGLISH/MANUALS/BRef/ATMT.html`.

[2] Python – user programmable block in python. `https://www.rexygen.com/doc/ENGLISH/MANUALS/BRef/PYTHON.html`.

[3] Rexygen how it works. `https://www.rexygen.com/rexygen/`.

[4] Zeungnam Bien and Jian-Xin Xu. *Iterative learning control: analysis, design, integration and applications*. Springer Science & Business Media, 2012.

[5] Lennart Blanken, Goksan Isil, Sjirk Koekebakker, and Tom Oomen. Flexible ilc: Towards a convex approach for non-causal rational basis functions. *IFAC-PapersOnLine*, 50(1):12107–12112, 2017.

[6] Frank Boeren, Abhishek Bareja, Tom Kok, and Tom Oomen. Frequency-domain ilc approach for repeating and varying tasks: With application to semiconductor bonding equipment. *IEEE/ASME Transactions on Mechatronics*, 21(6):2716–2727, 2016.

[7] Joost Bolder and Tom Oomen. Rational basis functions in iterative learning control—with experimental verification on a motion system. *IEEE Transactions on Control Systems Technology*, 23(2):722–729, 2014.

[8] Joost Bolder, Tom Oomen, Sjirk Koekebakker, and Maarten Steinbuch. Using iterative learning control with basis functions to compensate medium deformation in a wide-format inkjet printer. *Mechatronics*, 24(8):944–953, 2014.

[9] D.A. Bristow, M. Tharayil, and A.G. Alleyne. A survey of iterative learning. *Control Systems, IEEE*, 26:96 – 114, 07 2006.

[10] YangQuan Chen and Kevin L Moore. An optimal design of pd-type iterative learning control with monotonic convergence. In *Proceedings of the IEEE Internatinal Symposium on Intelligent Control*, pages 55–60. IEEE, 2002.

[11] Martin Goubej, Jana Königsmarková, Ronald Kampinga, Jakko Nieuwenkamp, and Stéphane Paquay. Employing finite element analysis and robust control concepts in mechatronic system design-flexible manipulator case study. *Applied Sciences*, 11(8):3689, 2021.

[12] Martin Goubej, Sven Meeusen, Noud Mooren, and Tom Oomen. Iterative learning control in high-performance motion systems: from theory to implementation. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 851–856. IEEE, 2019.

[13] Tom Oomen. Learning in machines. *Mikroniek*, 6:5–11, 2018.

[14] Maurice Poot, Jim Portegies, and Tom Oomen. On the role of models in learning control: Actor-critic iterative learning control. *IFAC-PapersOnLine*, 53(2):1450–1455, 2020.

[15] Julius O. Smith. Transposed direct-forms. `https://ccrma.stanford.edu/~jos/fp/Transposed_Direct_Forms.html`.

[16] Nard Strijbosch, Lennart Blanken, Tom Oomen, et al. Frequency domain design of iterative learning control and repetitive control for complex motion systems. In *IEEJ International Workshop on Sensing, Actuation, Motion Control, and Optimization*, pages 1–2, 2018.

[17] Masayoshi Tomizuka. Zero phase error tracking algorithm for digital control. 1987.