CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Electrical Engineering
Department of Control Engineering

Design and implementation of an iterative learning control system for a dynamic plotter

Master's thesis

Bc. Marek Bečka

Supervisor
doc. Ing. Zdeněk Hurák, Ph.D.

2022

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Bečka Marek**	Personal ID number:	**434721**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Control Engineering**		
Study program:	**Cybernetics and Robotics**		
Branch of study:	**Cybernetics and Robotics**		

II. Master’s thesis details

Master’s thesis title in English:

Design and implementation of an iterative learning control system for a dynamic plotter

Master’s thesis title in Czech:

Návrh a realizace iterativního učícího řídicího systému pro dynamický plotr

Guidelines:

Design and realize electronic modules and software for a control system for the simultaneously developed experimental mechatronic device for planar motion control – a planar plotter. The required functionality of the control system is to make the printing head follow the required trajectory, which will be visualized by plotting. Unlike an ordinary plotter, the new experimental device will will have a heavy printing head, which will call for taking the dynamics of the whole device into consideration while designing a controller, otherwise the system will tend to overshoot or even oscillate. Furthermore, the whole task of reference trajectory following will be repeated a few times. Imperfectness in following the reference trajectory (regulation error) will be logged during the first run and reused in the next runs to generate a compensating feedforward signal (to be added to the output of the feedback controller). The technique is referred to as Iterative Learning Control (ILC) in the literature. Thus your goal is to design and realize a control system capable of implementing prototype ILC controllers.

Bibliography / sources:

[1] D. A. Bristow, M. Tharayil, a A. G. Alleyne, „A survey of iterative learning control', IEEE Control Systems, roč. 26, č. 3, s. 96–114, 2006, doi: 10.1109/MCS.2006.1636313.
[2] K. L. Moore, M. Dahleh, a S. P. Bhattacharyya, „Iterative learning control: A survey and new results', Journal of Robotic Systems, roč. 9, č. 5, s. 563–594, 1992, doi: 10.1002/rob.4620090502.
[3] M. Steinbuch a R. van de Molengraft, „Iterative Learning Control of Industrial Motion Systems', IFAC Proceedings Volumes, roč. 33, č. 26, s. 899–904, zář. 2000, doi: 10.1016/S1474-6670(17)39259-5.
[4] J. van Zundert, J. Bolder, a T. Oomen, „Optimality and flexibility in Iterative Learning Control for varying tasks', Automatica, roč. 67, s. 295–302, kvě. 2016, doi: 10.1016/j.automatica.2016.01.026.
[5] Y. Wang, F. Gao, a F. J. Doyle III, „Survey on iterative learning control, repetitive control, and run-to-run control', Journal of Process Control, roč. 19, č. 10, s. 1589–1600, pro. 2009, doi: 10.1016/j.jprocont.2009.09.006.

Name and workplace of master’s thesis supervisor:

doc. Ing. Zdeněk Hurák, Ph.D., Department of Control Engineering, FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **09.02.2021** Deadline for master's thesis submission: **04.01.2022**

Assignment valid until: **30.09.2022**

doc. Ing. Zdeněk Hurák, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Bc. Marek Bečka
January 2022

Acknowledgments

First of all, I would like to thank my supervisor, Zdeněk Hurák. He is the best mentor I could ever imagine.

Thanks also to Krištof Pučejdl, who built the plotter. He gave me a lot of valuable advices.

I am grateful to Loi Do for this template and also for companion in the lab.

Other thanks: Adam Kollarčík for the model, my girlfriend Lenka for the support and correction, my parents for patience and encouragement. Other anonymous friends for their long term undying support.

Abstract

This thesis is about development of electronics and control system for a platform for experimenting with Iterative Learning Control (ILC). This mechatronic device has a form of a planar plotter, which has an advantage of a nice visualization of repeating patterns. The thesis first deals with design of electronics for this plotter. For this purpose two different existing BLDC motor drivers are used, which both communicate over CAN bus. Two interfaces to Matlab/Simulink are developed, one using Target to a BeagleBone Blue singleboard computer and the other Kvaser Leaf USB interface card. The control algorithms are written in the Matlab/Simulink language, where I implement the ILC controller and other service functions. Several experiments are performed to ensure the correct functionality of the ILC controller by itself and model of the plotter. The ILC algorithm is then shown on the physical plotter, with a visualization of improving drawings. Brief discussion about a longterm stability of ILC concludes the thesis.

Keywords: Iterative Learning Control, Repetitive Control, Dynamic Plotter, CAN communication, BLDC motor

Abstrakt

Tato diplomová práce se zabývá vývojem elektroniky a řídicího systému pro platformu pro experimentování s Iterativním učícím se řízením (ILC). Toto mechatronické zařízení má podobu planárního plotru, jehož výhodou je pěkná vizualizace opakujících se vzorů. Práce se nejprve zabývá návrhem elektroniky pro tento plotr. K tomuto účelu jsou použity dva různé existující drivery BLDC motorů, které oba komunikují po sběrnici CAN. Jsou vyvinuta dvě rozhraní pro Matlab/Simulink, jedno pomocí Targetu do jednodeskového počítače BeagleBone Blue a druhé s užitím USB/CAN převodníku Kvaser Leaf. Řídicí algoritmy jsou napsány v jazyce Matlab/Simulink, kde implementuji ILC regulátor a další obslužné funkce. Je provedeno několik experimentů, které mají zajistit správnou funkčnost samotného ILC regulátoru a modelu plotru. Poté je algoritmus ILC předveden na fyzickém plotru s vizualizací zlepšujících se výkresů. Diplomovou práci zakončuje diskuse o dlouhodobé stabilitě ILC algoritmů.

Klíčová slova: Iterativní učící se řízení, Opakující se řízení, Dynamický plotr, CAN komunikace, BLDC motor

Contents

1 | Introduction

This thesis focuses on design and implementation of electronics and control system for a planar plotter. The ultimate purpose of this build is to have a platform for developing algorithms of Iterative Learning Control (ILC). The intended goal is to see the improvement of trajectory tracking over repeated iterations, which is what the ILC should provide. A great advantage of the plotter will be the visualization of these improving patterns made by a marker on plexiglass. This could work as a nice visual demonstrator of ILC even for the wide public.

The results are documented in the thesis, by a video and the code is published in a Gitlab repository[1]. All the code is commented and organized in folders, the experiments are repeatable since the data are recorded and stored in the corresponding folders.

1.1 Objectives

The objective of the thesis is threefold. Firstly, I need to choose a platform, that suits well the requirements - it should drive a motor and handle the control computation. A natural division of these roles is to have a hardware driver of BLDC motors and a computer for the high-level control part. An interface needs to be developed between these parts.

Second task is to develop the control software itself, with a special focus on the Iterative Learning Control. There should be an option to visualize the drawing in a Software, so a tuning of the controller can be easily done. The ILC has a purpose of learning from past errors, which implies the need for a memory that stores the regulation errors. I assume the use of Matlab, which is a standard in a automatic control industry.

The last task is to merge the first two parts and test them on the hardware prototype of the plotter. The device was being manufactured simultaneously to the development of the electronics and software, which is made within this thesis (see visualization on Figure 1.1).

[1]`https://gitlab.fel.cvut.cz/aa4cc/dynacorexy`

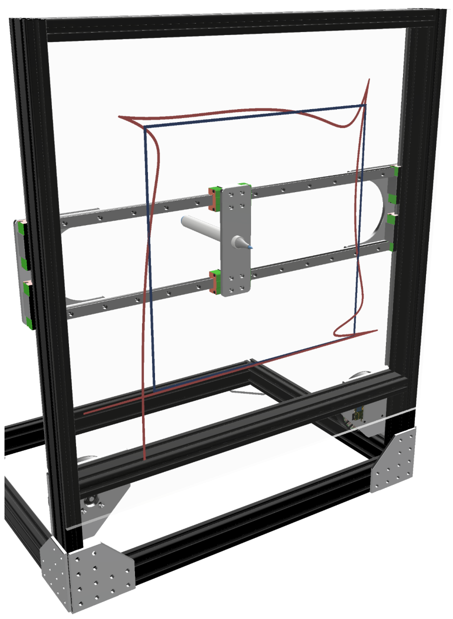


Figure 1.1: Visualization of the proposed prototype

1.2 Motivation

Many systems in our world repeat the same tasks over and over again. From a production line in a factory to a hockey player who shoots on a goal a thousand times in his life. However, there is a difference in these two examples - while the hockey player takes advantage of the repetition and tries to improve himself, the control systems in robots are typically tuned when created and stay the same. Iterative learning control makes a notion of this fact and tries to take advantage of it. It could be implemented on a robot in a factory, which performs same movements many times, or an autonomous car going around the same curve every day. Specifically, ILC is based on an idea that the performance of a system can be improved, if we can store errors from the past, process them, and use the information in the future.

1.3 ILC in Control Theory

In the language of control theory, ILC is a learning feedforward controller. The advantage is, that ILC can be used with a usual feedback controller. The role of the feedback in this setup is to suppress the nonrepetitive disturbances (e.g. noise, random disturbance), while the ILC takes the repetitive control task (reference tracking, or repeated disturbance). The repeated disturbance in this scenario could be for instance increased friction in one place of the motion due to wear and tear. ILC has a memory, so it can predict a coming change of the reference/disturbance and react in advance, unlike the traditional feedback, which always lags. As the name suggests, in case there is no repetition in the control challenge, ILC has no benefit and cannot be used at all.

1.4 Related Work

Quite a bit has been written about Iterative Learning Control, Repetitive Control and similar topics. From industrial solutions to more of budget proof-of-concept approach. Focusing on the printers and similar mechatronic devices, one name really stands out. It is Tom Oomen from Eidhovan university. One of the relevant papers is *Design Techniques for Multivariable ILC: Application to an Industrial Flatbed Printer* [1]. He has not focused only on pure academic research, but also on more educational format of the same theme [7]. In the lowcost "do it yourself" approach, this author has still something to offer - namely his recent series of customizing a consumer printer[2].

The main paper, which I will be following, is however the Survey on iterative learning control, repetitive control, and run-to-run control [11]. There is also an older survey of the same kind [6].

[2]https://www.maxvanmeer.nl/printer-project/part-1-requirements-preferences-and-constraints-low-cost-real-time-closed-loop-control-of-a-consumer-printer/

2 | Hardware

Because the development of this project was not straight forward, first let me briefly describe it.

The base design of the plotter has been proposed by my colleague Krištof Pučejdl, who also built it. All I had to do was small finishing touches like soldering and printing head customization, so it better holds a marker. Later, I also redesigned some parts for the purpose of artificial disturbance introduction. This will be described in this chapter.

Secondly, I need to discuss the motor driver issue. At the beginning I chose Ben Katz design, which our department already had a good experience with in a SK8O robot[1]. However, the combination of bad parts availability due to Covid-19 and unfinished development of the driver by Bc. Petr Brož[2] forced me to switch the driver to an Odrive. This change was done rather late in the project, which means a substantial volume of work is duplicated. However, since both the drivers were successfully finished and tested from Simulink, I will depict both of them for potential future use.

2.1 Plotter Description

Since our plotter has not been described anywhere before, let me do an overview of the construction and mechanics of it.

Traditional printers use Cartesian mechanics, where the printing head is moved by one motor in x and the other in the y coordinate. That has one inherent disadvantage - at least one motor is moving, which increases the inertia of moving parts. Higher inertia means lower possible accelerations, which is not desirable. If there is a requirement for the highest speed possible, this construction cannot satisfy our needs.

The basic idea of our plotter is to have motors stationary, mounted to the nonmoving frame of the printer. H-bot mechanical system allows just that, since forces are transferred with a system of belt and pulleys to a gantry and printing head. On the gantry, there are four pulleys and it can only move up and down (relative to the plotter), while the printing head moves only left and right (relative to the gantry)[3].

[1] https://control.fel.cvut.cz/nasi-doktorandi-postavili-robota-balancujiciho-na-dvou-nohach-budou-ho-ucit-skakat-do-schodu

[2] https://github.com/ptrbroz/motorcontrol_stm32g474re

[3] An animation of the H-bot can be found i.e. here https://www.youtube.com/watch?v=IkM2K7CsiHo

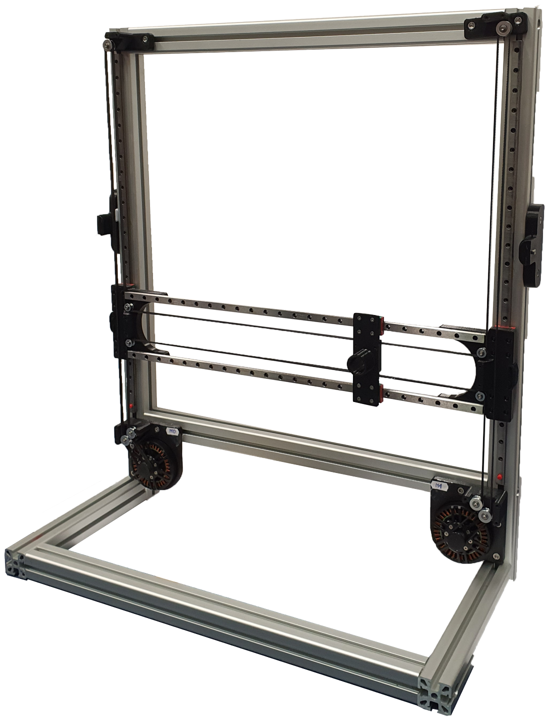


Figure 2.1: Real Plotter - overall view

This experimental device will be driven by BLDC motors. Note that in most scenarios, printers or plotters use stepper motors, because they are precise and easier to control. However, we are not so much trying to develop “the best plotter”, but rather a development platform for control algorithms - in which BLDC motors fit better thanks to their greater dynamics.

2.1.1 Kinematics

The kinematics of the H-Bot mechanism is formulated in eq. 2.1. The x and y are coordinates of the head, $\theta_{1,2}$ are angles of motors in radians and r is a radius of motor pulleys (fig. 2.2). For completeness and to avoid any confusion, I will also mention the coreXY system, which we do not use. CoreXY uses a little different belt design and kinematics differ in a sign, but the key principle remains the same.

$$\begin{aligned} x &= \frac{r}{2}(\theta_1 + \theta_2) \\ y &= \frac{r}{2}(\theta_1 - \theta_2) \end{aligned} \tag{2.1}$$

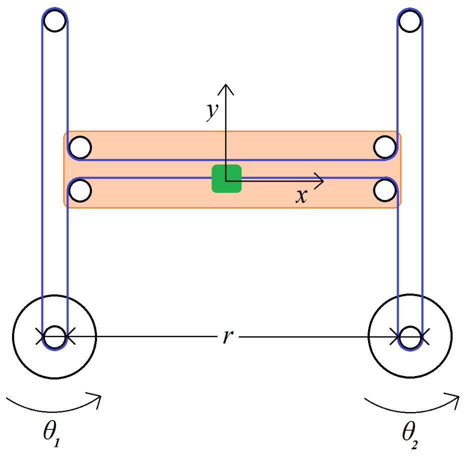


Figure 2.2: H-bot coordinate system

2.1.2 Disturbance Parts

One of the advantages of ILC is the ability to suppress a repeated disturbance. In order to test this, we need to perform experiments with a reliable disturbance inserted every iteration at the same spot of the drawing. I chose to artificially increase friction in a small area (around 30 mm) along the y axis of the plotter, which simulates eg. wear of the machine in a real world.

For this purpose, I modified the xy-coupling part, so it has a small bump. This bump then interacts with a stationary part mounted on a frame of the plotter. Both the parts are 3D printed, and the contact area is covered with velvet cloth. That increases the friction, reliability, and repeatability of the disturbance.

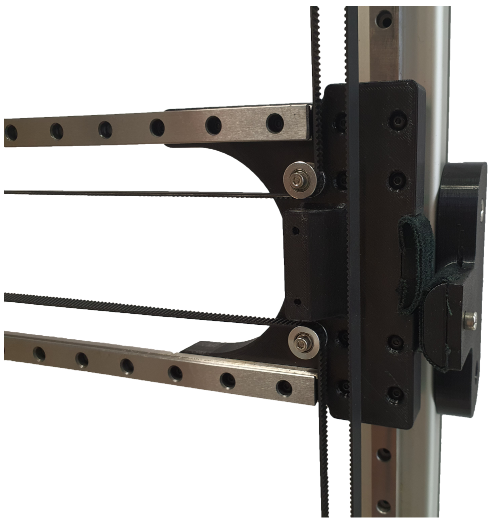


Figure 2.3: Disturbance mechanism in detail

2.2 Motor

In the plotter there are two brushless DC motors used, type eX8108 105KV. These were originally designed for RC quadcopters, but also have been proven in robotics. The peak power output is 750 W while having compact dimensions of 87 mm x 25 mm and 230 g of weight.

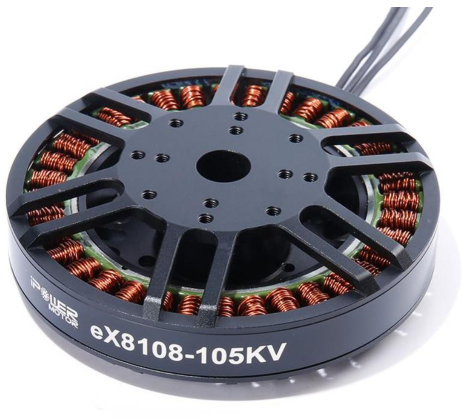


Figure 2.4: eX8108-105KV brushless DC motor

2.3 Ben Katz Motor Driver

This is a driver that was not used in the end. For the one that was deployed, see section 2.4.

The plotter was originally intended to use a motor driver first developed by Ben Katz for a Mini Cheetah on MIT [9] [4]. It is a 40 kHz 3 phase controller for BLDC motors. There is an ongoing development of this driver on our faculty. The main innovations are a new CPU STM32g474re, better three phase driver chip, and also CAN FD (Controller Area Network Flexible Data-Rate) communication protocol. However, I was provided the original version of a driver, designed by Ben Katz. The only difference from a user's point of view should be the use of older CAN (Controller Area Network) communication. That means some additional work probably needs to be done to adapt adequately to these changes. Otherwise one driver with one motor has been tested, using BeagleBone Blue as a Target from Simulink.

2.3.1 Control Computer

As the main brain of the plotter, I chose a popular single board computer BeagleBone Blue. It has an integrated CANbus receiver and transceiver, which means we can communicate with the motor driver with no other components needed. Note that some development boars have CAN support as well, but they are not able to transceive CAN messages without additional chip (e.g. BeagleBone Black).

Another big advantage of BeagleBone Blue is integrated Matlab support in form of a *Simulink package for BeagleBone Blue*. This package implements many of basic functions of the board, including basic things such as GPIO LED blinking, integrated H-bridge motor control, and an encoder. Supported are both compilation and deployment, or compilation and running in external mode.

However, the CAN bus support is not natively implemented. Therefore I had to develop a driver for this purpose. For a clear distinguishing between the physical driver board by Ben Katz, high-level control software and this low level Simulink BeagleBone support with CAN packet decoding, I will call it firmware.

2.3.2 Firmware

As mentioned earlier, I will use Matlab and Simulink for basically all control algorithms that need to be developed. Therefore some interface between the BeagleBone and Simulink needs to be set up. For this purpose, I used Matlab System Object, which is a powerful specialized object, which can among others call external C functions from custom written C code. This code is compiled using Matlab Coder and then deployed on our BeagleBone. In the C language, I will write my CAN bus communication and all the decoding arithmetics of the packets.

Behaviour and appearance of the system object block are controlled by the `.m` file with the corresponding name, in my case `motor_can.m`. In this file, it is defined input and output port number and types, parameters, and C function calling logic. The design was done with an emphasis on the reusability of the driver, even or other projects or applications.

I split the C code into two files, namely `motor_can.c` and `communication.c`. Header files of corresponding names are also made. My code uses SocketCAN driver, which provides an interface similar to Internet Protocol via the sockets. For the communication we need to do several steps, similar to TCP/IP communication - first the socket needs to be initialized and created, then the application will bind this socket to an existing can interface. After that, we can use read from and write to the socket.

2.3.3 Functions of motor_can.c

Although the purpose of all the functions is well documented in the code, for clarity let me sum it up here. I will omit the input and output arguments, as it will only make the text excessively long and therefore less clear. For more details, the reader can always check the code itself.

Function `print_debug()` is for debugging purposes only. The debug messages can be activated by `#define PRINTDEBUG` statement at the top of the file, which then ensures the `print_debug` function is called every time `DEBUG(‘‘debug message’’)` is called.

`can_setup()` first initializes the error output file, if the user wishes to (via the procedure mentioned above). The main function however is to open and configure the can socket and associate it to the defined can interface. The last functionality calls enter_motor_mode function, so the communication with specified can ID can begin.

int `enter_motor_mode()` sends a special command 0xFFFFFFFC for the motor to enter its operating “motor” mode.

Similarly `disable_motor_mode()` deactivates the motor by sending 0xFFFFFFFD command to it.

All the can messages are sent via the `send_can_frame()` function.

Function `input_reference()` takes all the inputs (position, velocity, Kp, Kd, feed forward current) and sends them to the motor. It also receives the answer and unpacks it using `unpack_motor_measurement()` to the global variables of respective names.

Functions `return_current`, `return_velocity`, and `return_position` are simple getters of respective values, with an exception of the latter one. The `return_position` also checks for potential discontinuity, and patches it. The discontinuity appears when a motor encoder crosses its origin position.

Finally, the `can_terminate` first disables the motor mode calling `disable_motor_mode` function. It also closes the socket and connection, and the file descriptor of debug file if it was created in the first place.

2.3.4 Functions of communication.c

Similarly to the main C file `motor_can.c`, I will briefly mention the functions of a communication file. As the name suggests, it has the purpose of packing and unpacking the messages of can communication. Note that the template is available in **??**, but it is written in C++. Since I had no success making C++ compilation work with Matlab Coder and because of some custom requirements, I had to rewrite it in C.

There are two main executive functions in the communication. The first is the `pack_motor_reference`, which takes float numbers from the Matlab and translates them into a can message. Secondly, `unpack_motor_measurement` translates the answer from the motor and translates it into a floats, which Matlab can work with. Both the functions call the lowest level bit operators from header files.

These also have their counterparts, `unpack_motor_reference`, and `pack_motor_measurement`, which exist only for help with development purposes, because the hardware was not available all the time due to covid situation.

2.3.5 Simulink Driver Function & Verification

Unfortunately I do not have any data of simple experiments, that could show driver functionality. Right now the hardware is not available anymore, but I have one video of the motor following rectangular waveform position reference[4].

Let me explain the basic functionality. The user can set five inputs, the motor driver receives all of them and combine all of them together to get one command for the motor. The values are position and velocity inputs, which are set as a reference in a control loop with PD regulator with customizable coefficients Kp and Kd. Tested values of the constants with no load and decent behaviour are i.e. $Kp = 3$ and $Kd = 0.3$. If we do not want to use any position or velocity reference, we have to set these values to zeros (the zero position and velocity are not enough, it would cause the motor to move to zero). On the output, the motor replies with actual position, velocity, and current flow. All values are a float type (single precision).

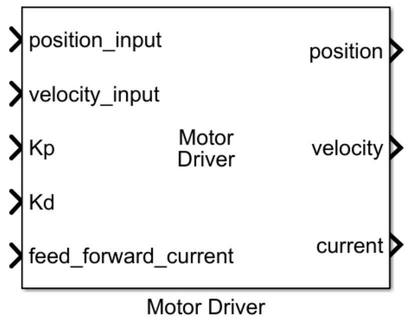


Figure 2.5: Simulink block of motor driver

2.3.6 Motor Identification

Since the motor driver returns the information about an actual current floating it, and I can also set a reference current, I performed an experiment in purpose to obtain a transfer function of these two. I've focused on the current because the assumption is that I am going to use it in my ILC control experiments to have more control over the motor, rather than using integrated reference/velocity controller.

For the identification purpose, I first set all commands to zero, varying only the feed_forward_current input. I choose PRBS - pseudo random binary sequence with defined bandwidth and sampling frequency, generated using `idinput` function of Matlab. I generated 3 sets of 120 second long data, two for model fit and one for verification. This signal is then fed to the motor, which should ideally excite all its modes. The experiment of course was performed over the CAN communication, which had a frequency of 1 kHz. That with the noisy data has

[4]https://youtu.be/J_v39mfhuxk

emerged as a likely limiting factor, as almost no dynamics can be observed and extracted. Using System Identification Toolbox I obtained a discrete 4th order model with an 80% fit to the testing data. Equation of the transfer (rounded to 4 decimal places) is on eq. 2.2 and comparison to the data (only a fraction of the data) can be seen in the fig. 2.6.

$$H(z) = \frac{-7.90e - 5z^2 + 0.0348z - 0.0365}{z^4 - 1.341z^3 + 0.3868z^2 - 0.0278z - 0.0021} \tag{2.2}$$

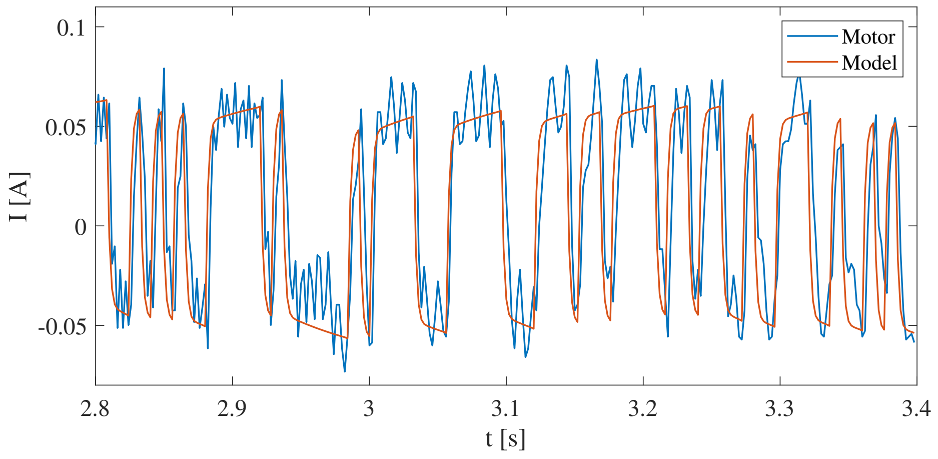


Figure 2.6: Model and motor response comparison

2.4 Odrive Driver

The final version of the plotter runs on Odrive v3.5 (fig. 2.7), which is an open source control board for BLDC two motors. It supports a peak current of 120 A per channel. The board can be commanded via serial, USB or CANbus. I chose the CAN. It has the arguably the best speed, latency and reliability, plus I had some development experience from the Ben Katz driver.

The implementation to Simulink is different from the other driver though. I took advantage of good documentation of CAN communication, including a .dbc file from Odrive website[5]. This file defines the data transmitted within a CAN frame, including names, scaling, offsets and other parameters. All this was programmed manually in C in the previous Ben Katz driver. I edited this file so that only one .dbc file defines communication with both motors, in Odrive terminology Axes.

For the Odrive-Simulink connection, I took advantage of Vehicle Network Toolbox, which supports among others Kvaser Leaf Light v2 card. With this USB-CAN interface card Simulink running on PC can command the Odrive. Odrive support a wide variety of control

[5]https://docs.odriverobotics.com/can-guide

inputs. The control structure (fig. 2.8) allows e.g. running the feedback loop on Odrive and feedforwarding velocity or current, which can provide ILC.

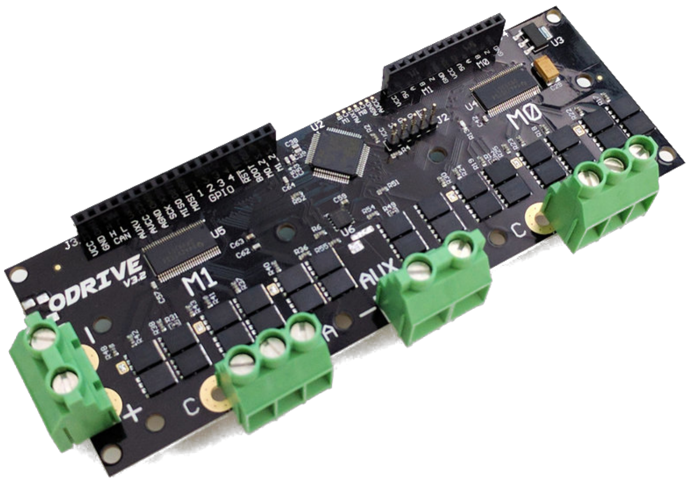


Figure 2.7: Odrive motor driver

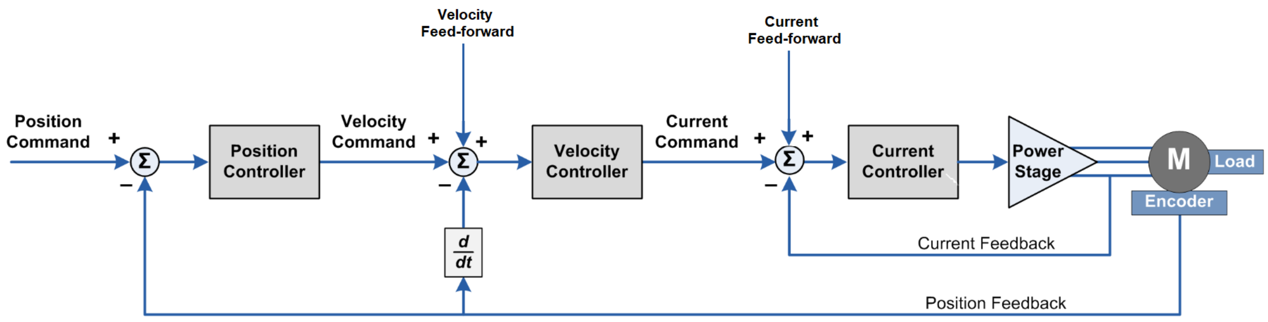


Figure 2.8: Odrive - integrated control structure

2.4.1 Odrive Setup

Before use, Odrive needs to be configured. That includes specifying motor and encoder type, limiting the current etc. It can be done by connecting the Odrive via USB to PC and using odrivetool[6]. The Odrive mounted on the plotter is already configured, but if there is a need for a board change in the future, all the configuration commands I have used are written in `\code\drivers\kvaser_odrive_driver` folder.

2.4.2 Odrive Driver in Simulink

The `Odrive commanded with Kvaser` block is part of the main `plotter_lib.slx` library. In order for the simulation time to correspond with "real time", pacing has to be enabled to

[6]`https://docs.odriverobotics.com/odrivetool`

slow it down. Note that unfortunately, this solution does not guarantee response times and thus does not make the simulation true real time. That will be briefly discussed with an experiment in Chapter 5.2.2. The whole driver has three main parts, namely configuration, CAN trasmit and CAN recieve.

CAN configuration blocks first setup the channel with defined speed and define the interface (in our case Kvaser Leaf Light v2). In this group, there is also initialization logic, which send input mode, control mode and axis state. Terminate block (fig. 2.10) does the opposite - after the experiment ends the axes can relax if the user chooses so. That can be specified within the parameters file.

CAN transmit consists of can pack blocks, which pack raw data into messages, which are sent via canbus. Position is modified by `axis\_offset constant`, which compensates the startup to home position offset.

CAN receive (fig. 2.9) contains an eponymous block, which upon a received message triggers can unpack logic. After the reverse position modification, the values are passed a level up.

For demonstration purposes, only two example Simulink schemes are attached. For more detailed documentation, user can refer to the Simulink library and to Gitlab.

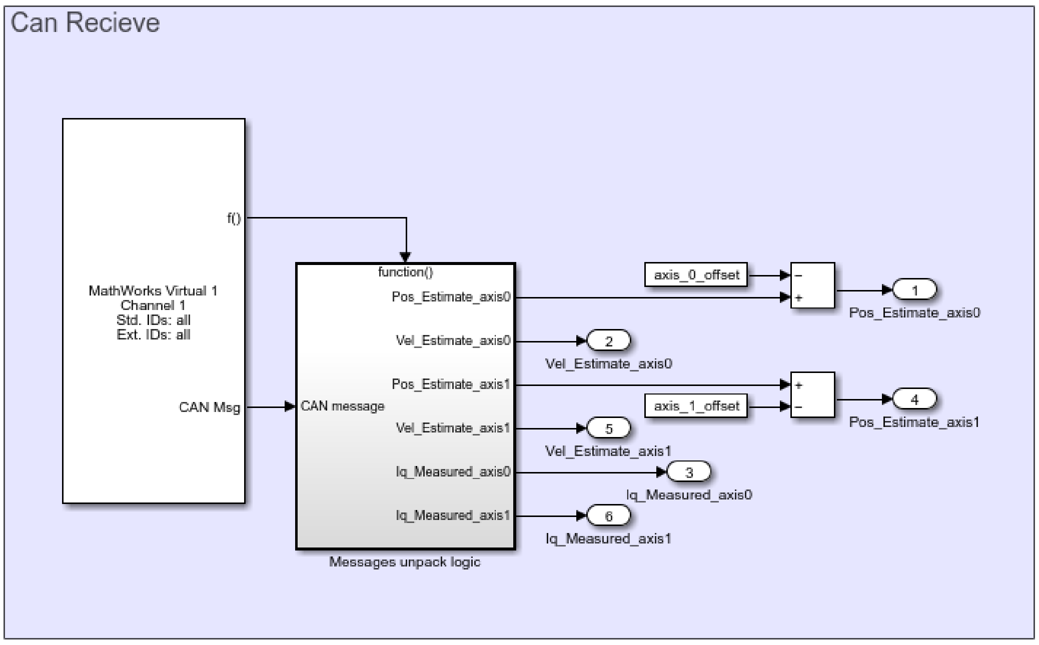


Figure 2.9: Odrive driver - recieve logic

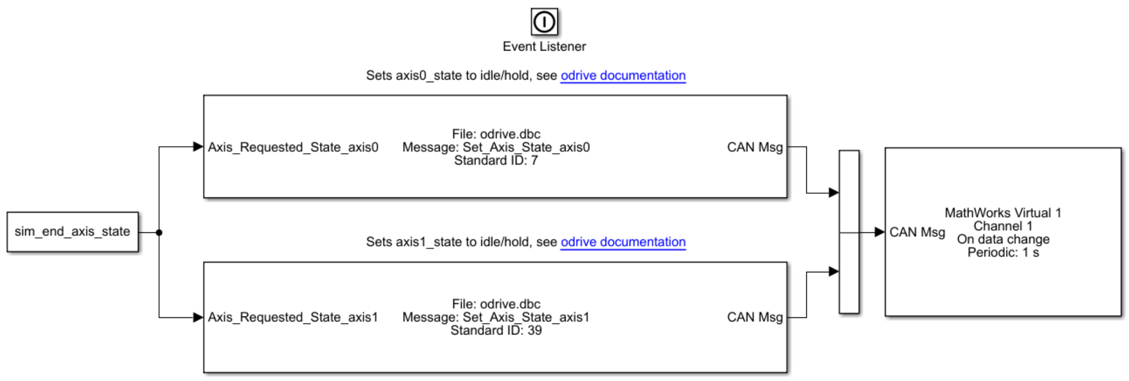

Event Listener
Sets axis0_state to idle/hold, see odrive documentation
Axis_Requested_State_axis0
File: odrive.dbc
Message: Set_Axis_State_axis0
Standard ID: 7
CAN Msg
sim_end_axis_state
Sets axis1_state to idle/hold, see odrive documentation
Axis_Requested_State_axis1
File: odrive.dbc
Message: Set_Axis_State_axis1
Standard ID: 39
CAN Msg
CAN Msg
MathWorks Virtual 1
Channel 1
On data change
Periodic: 1 s

Figure 2.10: Odrive driver - terminate logic

3 | Control Software

3.1 Overview

As mentioned briefly in the introduction, Iterative Learning Control is a type of feedforward regulator, which learns and takes advantage of previous iterations of the same task. It stores information about the regulation error and processes it in the next iteration to improve the signal. This implies the first design requirement, namely that the ILC controller must have memory and information about the iterations. I designed the ILC setup in such a way, that we have access to a time sample k in the iteration and also total iteration number j, so it is easier for further development to implement even a higher-order ILC algorithms, which use information about the error evolution over more that one iteration from the past.

Let me state the basic math symbols used in relation to the ILC. Mostly, I will be following the notation of the Survey [2], where k is the time index, j is the iteration index, w_j is the ILC control contribution, u_j is the plant input, y_d is the demanded tracking reference, y_j is the output and d is an exogenous signal which repeats each iteration. L is the learning function of ILC, Q is the output filter of ILC. C is the standard controller in the feedback loop and finally, G is the plant.

I will use two configurations of ILC setup. The parallel version, which is in classic feedforward configuration, can be seen in the fig. 3.1. I will use parallel setup in further analysis in this chapter and also in experiments with the model. However, in practical experiments with the real plotter, the serial version (fig. 3.2) turned out to be better, so I am mentioning it here for comparison and reference.

Each iteration, the ILC algorithm uses the control error of previous iteration $j-1$ and updates signal l_j via the matrix L. The L is usually a P or PD learning function since I term is naturally achieved through the integration of errors over the trials. For a PD learning function, we can write equation 3.1

$$l_j(k) = k_p e_{j-1}(k+1) + k_d(e_{j-1}(k+1) - e_{j-1}(k)), \tag{3.1}$$

where k_p is the proportional gain and k_d is the derivative gain.

The l_j is then added to the previous $w-1$ and filtered with Q to get

$$w_j(k) = Q(w_{j-1}(k) + l_j(k)) \tag{3.2}$$

Q is usually a low-pass filter, which has the purpose of disabling learning at high frequencies. That is desirable for added robustness, noise filtering and it could also help to achieve monotonic convergence. [2, p. 102]. Some ILC algorithms do not use Q filter, which is requirement for perfect tracking. The downside is that it can lead to bad transient behaviour.

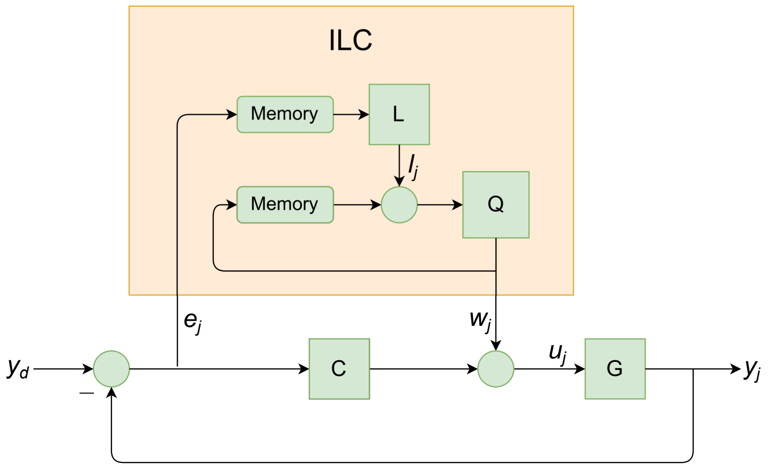


Figure 3.1: ILC - parallel setup

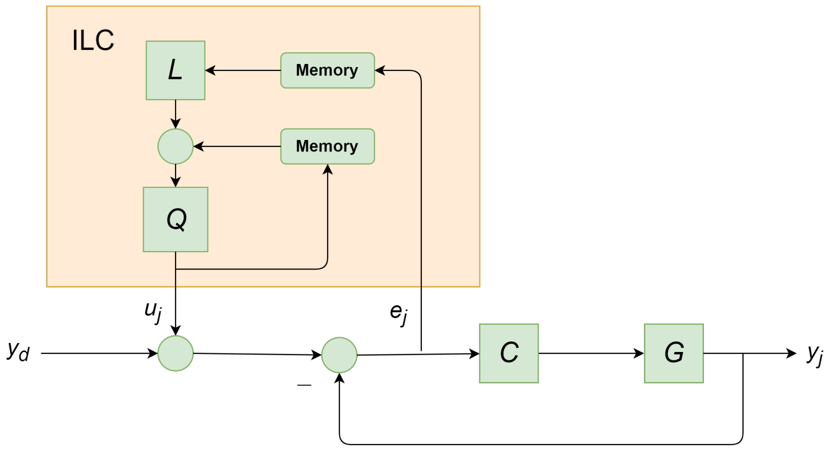


Figure 3.2: ILC - serial setup

3.2 Matlab Structure

All the parameters are in the `library_parameters_reference` folder. Here are two main files, `model_parameters.m` and `real_plotter_parameters.m` where the reference is generated, you can specify a sample time, parameters of the model, and all other simulation parameters. All the blocks and experiments are properly parametrized, so if you change any value here it should not break the consistency.

For the purpose of comfortable usability of all the Simulink blocks, I created a Plotter library called `plotter_lib.slx`. It contains a bunch of blocks, some of them being a subset of another. I will describe the blocks in more detail later in this chapter. The most important ones are

- Model of the plotter, unchanged original from my colleague
- Reduced model of the plotter (for $\mathcal{H}_\infty$ controller design), which lacks the integrated outputs
- ILC_source, which generates the reference and control signals for ILC block
- ILC_v2, which is the block for iterative learning control regulator
- Quadratic evaluator for the performance evaluation

In the same library, there are also two drivers of the BLDC motor (Sections 2.3, 2.4).

- Ben Katz commanded with BeagleBone Blue
- Odrive commanded with Kvaser

3.3 ILC Implementation

3.3.1 ILC Reference

Considering the previous, first I decided to create a dedicated Simulink block of reference and associated control signals. As parameters, the block takes an array of a single iteration of reference and a sampling time. Those I have parametrized and defined in the workspace of Matlab. The block with its outputs can be seen in the fig. 3.3.

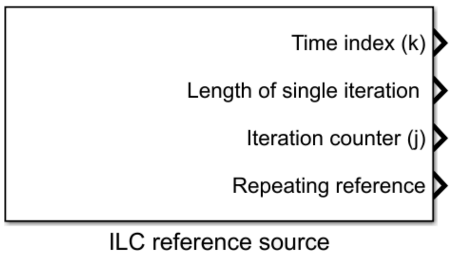


Figure 3.3: ILC reference Simulink block

The outputs have the following functionality - the *Time index k* is a number of sample, which resets every iteration. The *Repeating reference* simply takes the specified reference and repeats it over and over again. *Iteration j* is a number of the currently ongoing reference iteration, and lastly, *Length of a single iteration* output is actually a full signal of one reference iteration, which has the purpose of memory allocation in the main ILC controller.

3.3.2 ILC Controller

As a reference design and for verification, I designed a simple ILC implementing PD-type learning function (see eq. 3.2). It uses a memory two memory blocks, which stores information about previous iterations w_{j-1} and e_{j-1}. The coefficients ILC_{kd} and ILC_{kp} can be set within the parameters file. Inside the memory blocks, there is a simple Matlab function, which takes the sequence, copies it over, and sends it back to itself via the Unit Delay with length initialization. That is why I needed the length signal, to initialize a dimension of the $\frac{1}{z}$ memory.

With the real plotter experiments, the need for two more features emerged. The *can_delay* enables to fine tune the ILC with respect to the communication delays and *iter_count j* makes the learning disabled on the first run. That prevents a bad initial condition to have an impact on the performance. The ILC can be seen in fig. 3.3, with the inside in fig. 3.5.

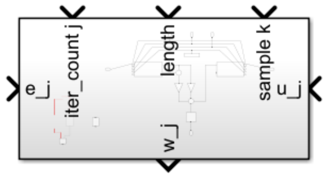


Figure 3.4: ILC Simulink block

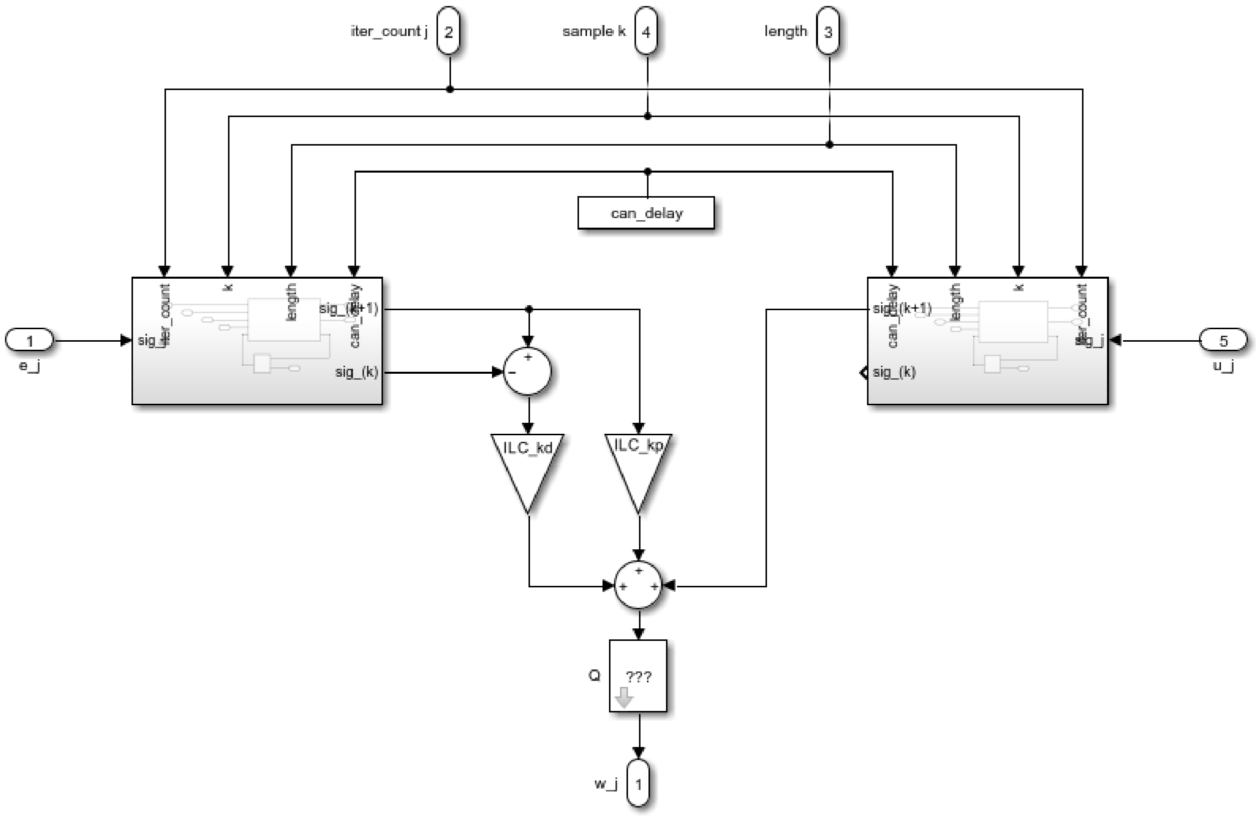


Figure 3.5: ILC Simulink block - inside

3.3.3 ILC Performance Evaluator

For the evaluation of the performance of each run, I designed a specialized block. This simple quadratic evaluator takes k and e_j, squares and sums up the control errors over an iteration. Then it holds this value on its output over the whole next iteration. This criteria is clearer and better for development, than evaluating it each step. Block is depicted in fig 3.8.

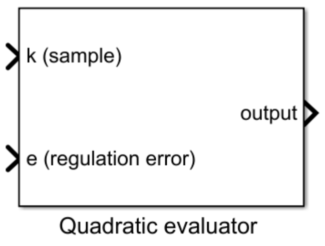
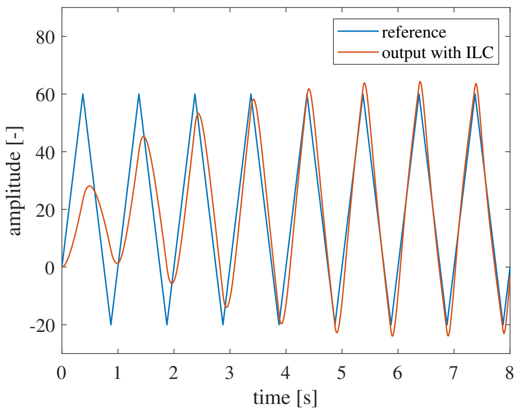


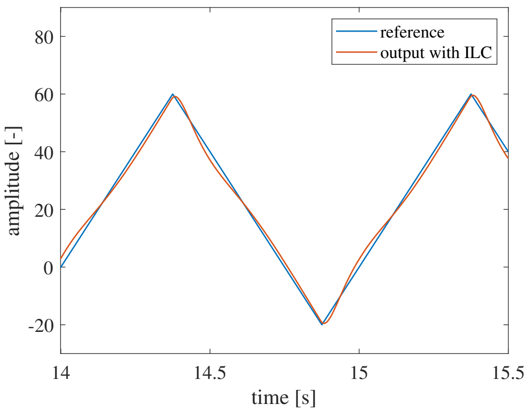
Figure 3.6: Quadratic evaluator

3.3.4 ILC Verification and Experiments

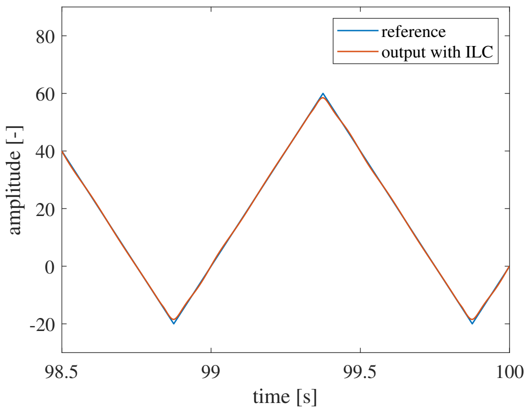
To verify both the generator of reference and other control signals and the ILC controller, I have done some experiments. First I used a SISO 1^{st} order system with one of the simplest transfer functions $G(s) = \frac{1}{s+1}$. For a feedback controller C a PD controller was used, with coefficients estimated at the first attempt to $K_p = 3$, $K_d = 0.1$. The learning function of the ILC was set similarly elementary to $ILC_{kp} = 0.1$ and $ILC_{kd} = 0$. A filter Q is a simple running mean, a window of 4 samples. The result can be observed in the fig. 3.7.



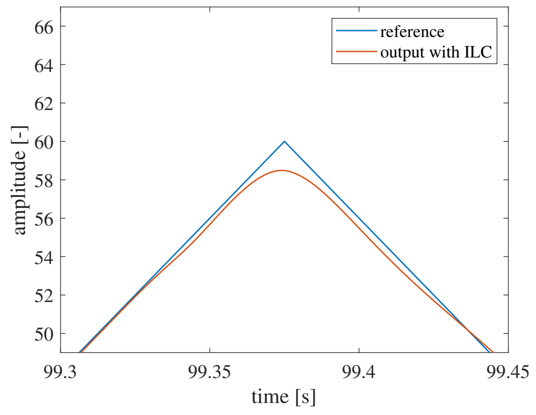
(a) First 8 iterations



(b) Around iteration 15



(c) After 100 iterations



(d) After 100 iterations - detail

Figure 3.7: ILC first experiment

As we can see, in this scenario ILC converges really nicely. In the beginning, the poorly tuned PD regulator lags and do not even closely reach the desired trajectory, but ILC and its feedforward action quickly takes over 3.7a. Around iteration 4 the tracking is good, but it still lags. For a small period (around 5 iterations) the ILC begins to slightly overshoot, but just around iteration 15, the tracking stabilizes 3.7b. Around iteration 100 the tracking is nearly perfect, as we can see on 3.7c or in detail 3.7d. You may notice, that the controller seemingly is not causal and predicts the future, as the output begins to decrease before the reference. That is the additional information from past iterations.

Let's see, how the ILC performs when the evaluation criterion is a e_j^2. Using the quadratic evaluator, we can observe a nice monotonic convergence. The curve is in fig. 3.8, please note the y-axis in a logarithmic scale.

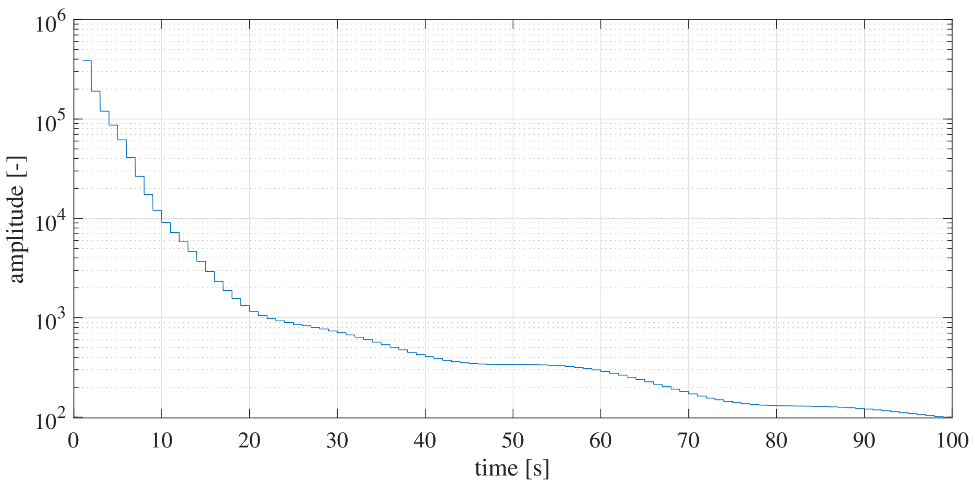
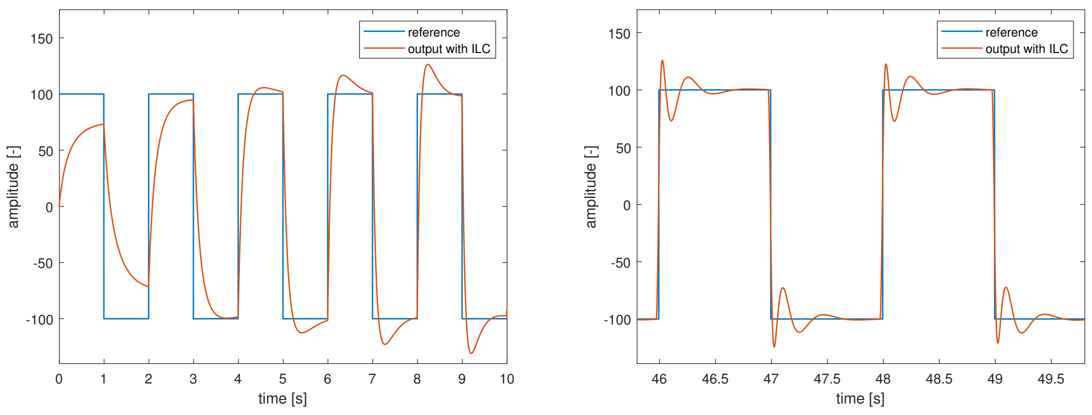


Figure 3.8: Quadratic performance, sawtooth reference

However, not with every scenario ILC works this smoothly, at least for a first try. Let's now try exactly the same setup, but a rectangular reference.

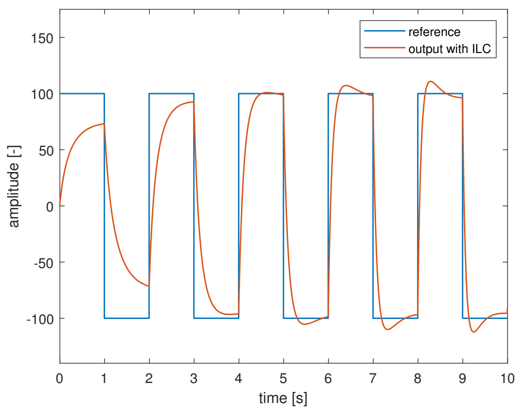


(a) First 5 iterations

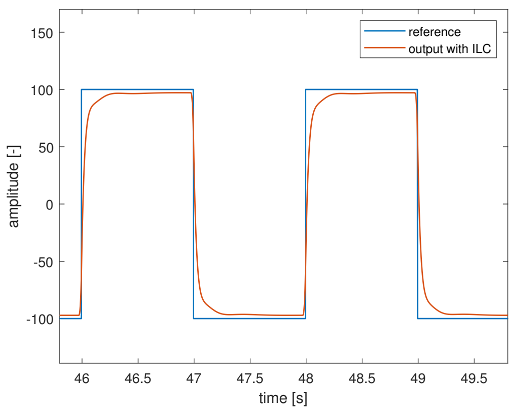
(b) Stady state after 25 iterations

Figure 3.9: ILC rectangle response, with moving average Q filter

The response looks alright only the first two iterations, then it overshoots way too much. And it does not get better, in fact, the quadratic error at one point even begins to rise. It looks like ILC tries to compensate for the error a little too much. However, adjusting the learning coefficient ILC_{kp}, or introduction of a derivative component does not help, it can only make the problem appear sooner or later. What we need is to tune the Q filter. I designed Q in Filter Designer utility, so it has $F_{pass} = 0\,\text{Hz}$, $F_{stop} = 20\,\text{Hz}$, and attenuation magnitude of 30 dB, so it has a order of 57. After these adjustments, our system produces following response (fig. 3.10)



(a) First 5 iterations



(b) Stady state after 25 iterations

Figure 3.10: ILC rectangle response, with advanced Q filter

On the performance evaluation, we can observe the non-montonic convergence of ILC with a moving average filter (fig. 3.11). At one point, the performance begins to be worse. With more advanced filtering, the performance curve looks better, although in absolute value in the steady state it is worse (fig. 3.12). That is because with the better filtering I effectively lowered the ILC contribution to the control. That is a price for suppressing the excessive overshoots.

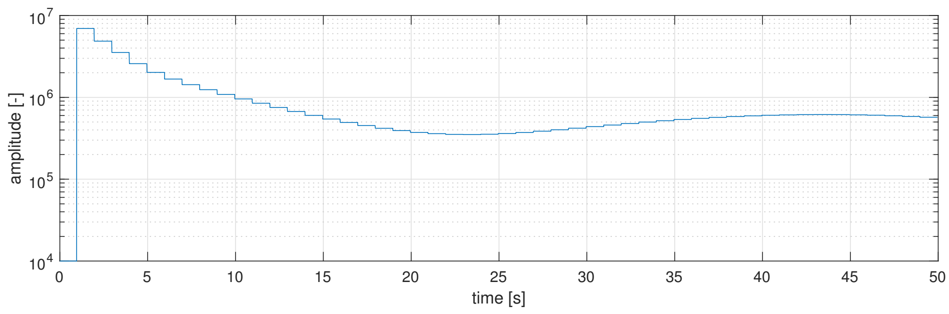


Figure 3.11: Quadratic performance, rectangle reference, moving average filter

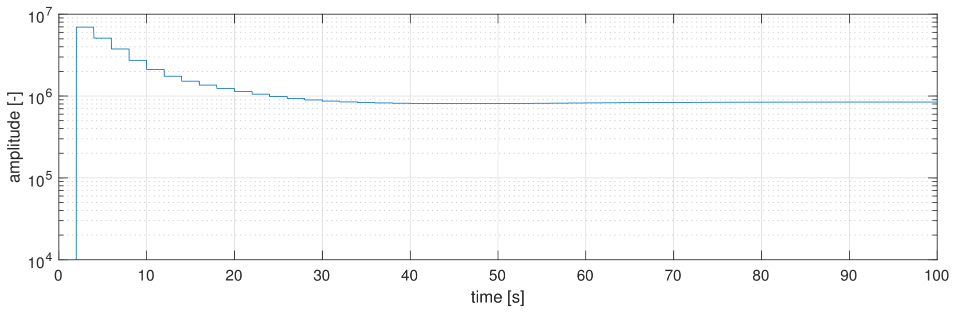


Figure 3.12: Quadratic performance, rectangle reference, advanced filter

4 | Experiments With the Model

4.1 Model Overview

At the time of writing this chapter, I had no access to the physical plotter due to hardware manufacturing delays. For the control algorithms development, I had to fully rely on simulation. Therefore I needed an accurate model of the physical plant, which I can rely on. I have been provided such a model by my colleague Adam Kollarčík, who provided me with a Simulink model developed with the use of bond graphs techniques. I want to declare, that I have **not** contributed to the model development in any way. My only task was to understand and verify the model or adjust the parameters to comply with my needs.

The model has two inputs $T1$ and $T2$, which represents torque input to the motor 1 and 2 in Nm. On the output side, we can observe the state of both the motors and the drawing head. Motors have $\omega_{1,2}$, head $v_{x,y}$. With their integrated counterparts $\theta_{1,2}$ and $position_{x,y}$ it makes a total of 8 outputs. Make note that in contrast, the currently proposed plotter has no measurement of the drawing head position. We can only observe the result - drawn shape - and use data from the motors on the “other side” of the belt.

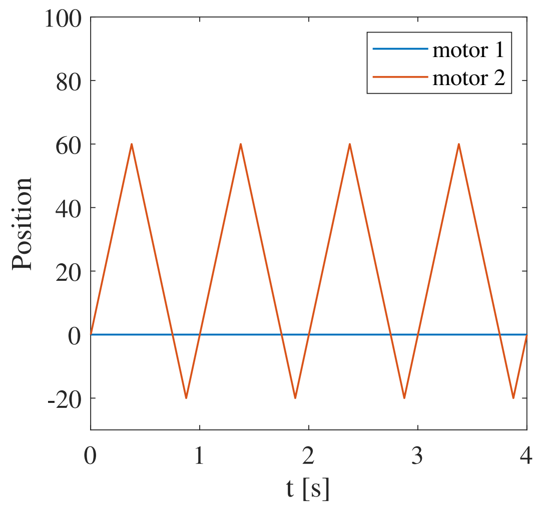
In the model, there are parameters for all the frictions, weights and compliances of all the individual belt sections. These parameters were estimated on the basis of this master thesis [9], that deals with h-bot modelling.

4.2 Model Verification

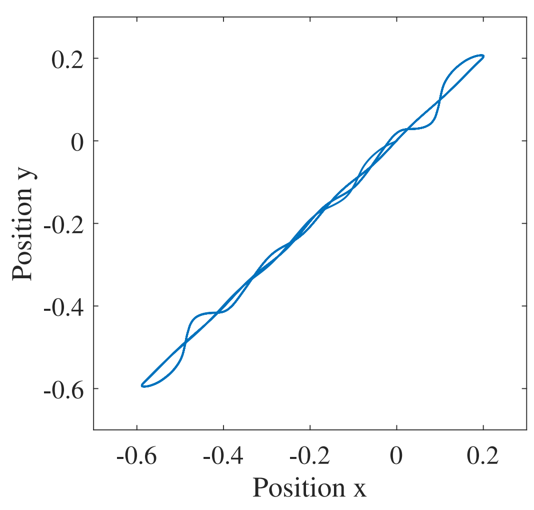
In this section, I will verify the model, see if it roughly fits the kinematics (eq. 2.1) and get feel of its dynamics.

4.2.1 Sawtooth Experiment

First of all, I had to develop some kind of a simple feedback control, so the drawing head can track a reference trajectory. For this purpose, I tuned a simple PID controller by intuition. For the first try, I tried one independent control loop per each motor from motor angle θ. I set $K_p = 3$, $K_i = 5$ and $K_d = 0.3$. With the feedback set up, fix one motor (set the reference to zero) and move only with the other one to draw a diagonal.



(a) Reference



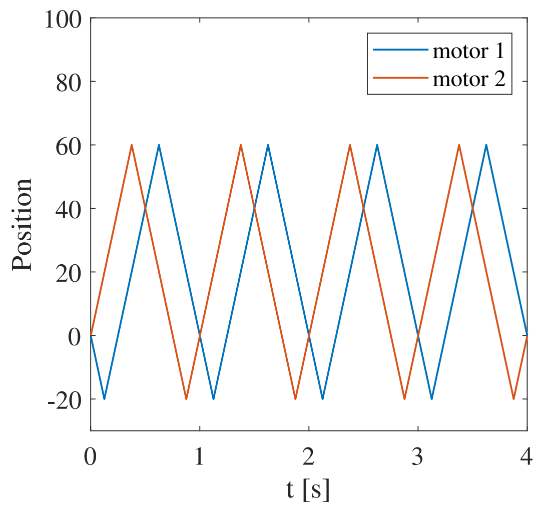
(b) Drawing

Figure 4.1: Diagonal drawing

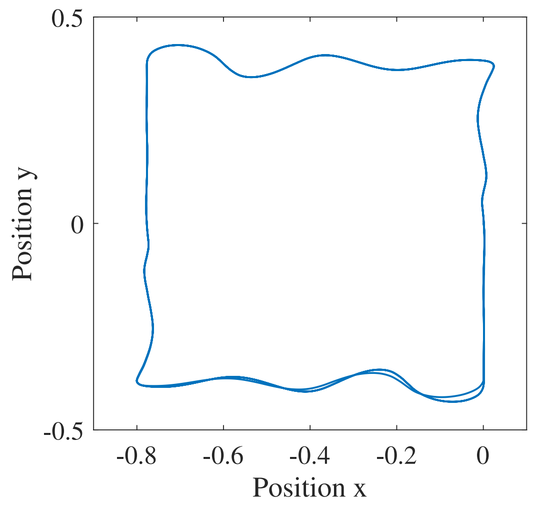
And indeed, the model behaviour seems to be reasonable. Here we can notice a slight cross-coupling between the motors, which will later be a challenge to deal with. This coupling originates from friction between the belt and pulleys on the gantry. Its magnitude is tunable in `model_parameters.m`.

4.2.2 Square Experiment

I chose the simple square shape as a general reference drawing path, due to its basic nature and also because of its sharp corners and the sequent control challenge. Square drawing with the same setup and paramaters $K_p = 3$, $K_i = 5$ and $K_d = 0.3$ is in fig. 4.2.



(a) Reference



(b) Drawing

Figure 4.2: Square drawing

We can see the head to draw a decent square with some overshoots. The overshoots seem to be much higher in the y direction than in x. That is because in `model_parameters.m` the weight in y direction is approximately 10 times higher than in x direction. That models the additional weight of the gantry and subsequent inertia causes a bigger overshoot. On the other hand, in x direction only the inertia of a light printing head plays a role.

4.3 Control Requirements

Before proceeding further, let me state some theoretical control requirements. Looking at the eX8108-105KV datasheet, max peak torque 3 N m seems like a reasonable requirement. The motor can safely deliver that when the speed is lower than 30 rad s^{-1}. Control loop 1 kHz is a verified value from previous projects and the BeagleBone can handle it with no problem, at least for one motor that I could test. Measured quantities are motor positions, angular velocities, and actual current.

However, that is the theory. Looking at the control action of conservative PID square drawing (fig. 4.2), the peak value of control signal u_j is over 10 N m, with the mean being over 5 N m. I briefly looked at it and ask my colleague with no clear conclusion. The reference is a small simple square drawn in $t = 1s$, which looks like a reasonable requirement every plotter should easily draw. The solution I decided for is to simply ignore this and do not look for the root of the problem, since the model parameters are estimated anyway and based on previous thesis of someone else. When we have the actual hardware, we can fine tune and verify the model and work with it.

Since on my model, it looks the control action is one order of magnitude higher, I choose my limit to be 10 times higher of $T_{max} = 30$.

4.4 ILC Experiments on the Plotter

4.4.1 Naive Implementation

The next step I tried is to implement my ILC block to improve the square drawing. Reference remains the same square (fig. 4.7a). Parameters of feedback controller and ILC are the same as in ILC verification experiments 3.3.4 (in fact they were tuned for this experiment in the first place). In this experiment, I approached the problem in SISO way, i.e. two independent control loops from θ to model input.

Parameters summary: feedback controller C with coefficients $K_p = 3$, $K_d = 0.1$. The learning function of the ILC has $ILC_{kp} = 0.1$ and $ILC_{kd} = 0$. For a filter Q i used the conservative advanced one, with $F_{pass} = 0\,\mathrm{Hz}$, $F_{stop} = 20\,\mathrm{Hz}$, attenuation magnitude of 30 dB. The result can be observed in the fig. 4.4.

The drawing cannot be described as anything other than a total failure. In the figure there

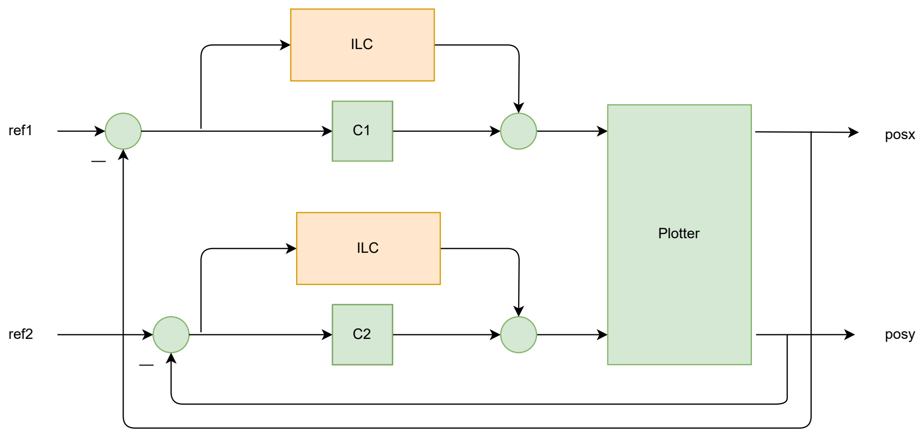


Figure 4.3: ILC implementation attempt

are only 6 iterations and the drawing already diverged to the point that it is not clear what the reference is. The control action diverges as well. Clearly, the crosscoupling in the model is strong enough to cause a fatal instability in our control loop. While trying to solve it, I tried both lower learning coefficients or more aggressive low pass filter Q. It does not solve the problem, it could only make it appear a little later.

4.4.2 MIMO Approach

Since single input, single output control strategy is insufficient for this scenario, the next step I decided for is a MIMO controller design. There is an extensive theory and tons of methods in this area of control, however, after consultation with my supervisor, I decided to focus on $\mathcal{H}_\infty$ family of methods.

$\mathcal{H}_\infty$ methods are used for obtaining the controller, which is in a mathematical sense optimal and can guarantee performance. $\mathcal{H}_\infty$ synthesis is the process, that calculates the controller that minimizes the gain between external input to error signal while using the least amount of actuator energy. The trick is to design the control problem, so an existing solver can be used. For this purpose, I designed a generalized plant, which contains a model of our plotter, the measured signal we have access to, inputs we have control over, and other signals we want to consider in this problem. All these signals are weighted, and the weights can be frequency dependent (i.e. can be treated with a transfer function). With regard to potential application implementation and because of the ILC augmentation simplicity, I decided to force a controller structure. In my case, I simply added two PID controllers to create a MIMO 2x2 controller (see fig. 4.5). Then I added performance weights to the inputs of the plant $Wu1, Wu2$, to output of the plant $Wt1, Wt2$ and to the control error $Wp1, Wp2$. The labels and procedure was done in compliance with [8].

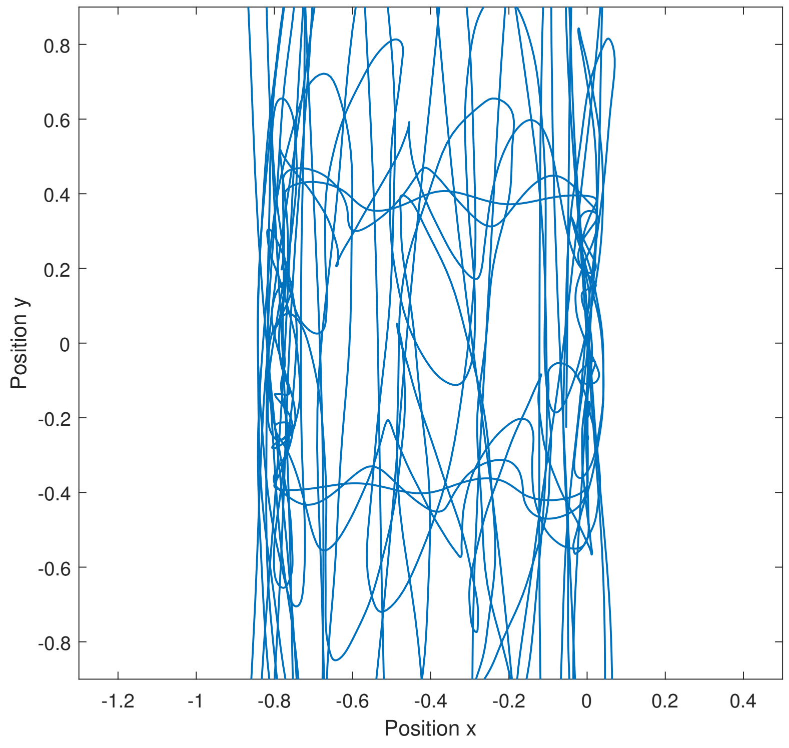


Figure 4.4: ILC implementation attempt

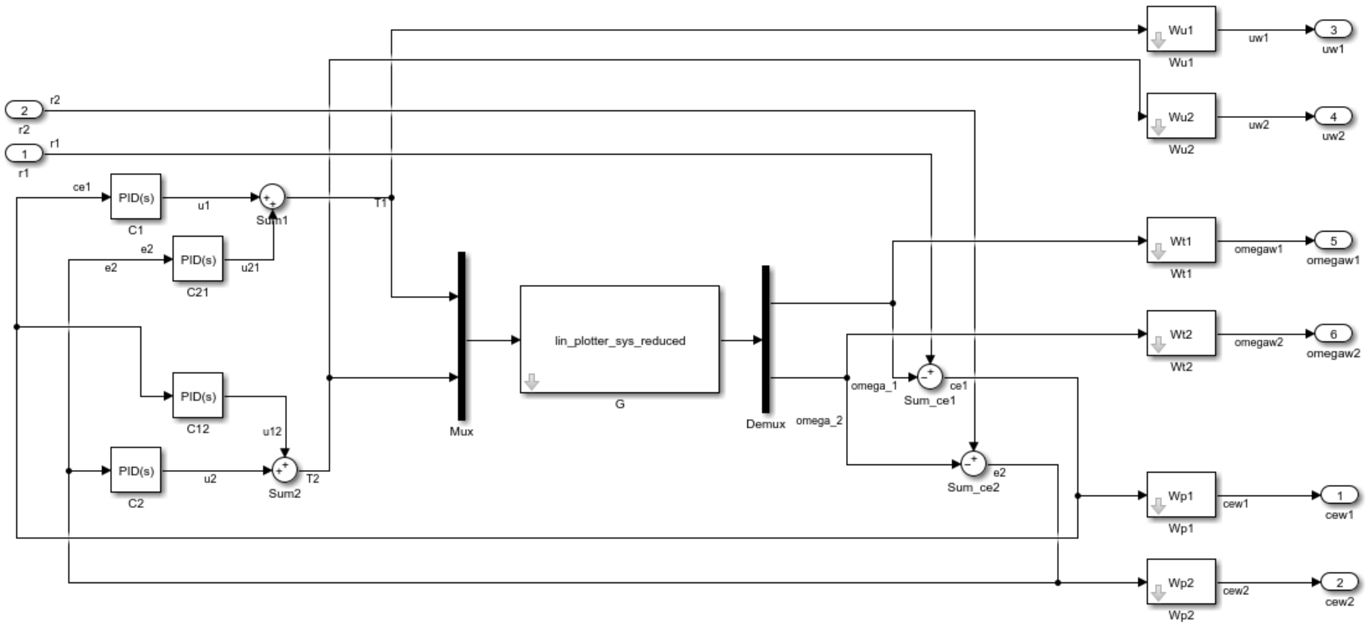


Figure 4.6: Generalized plant

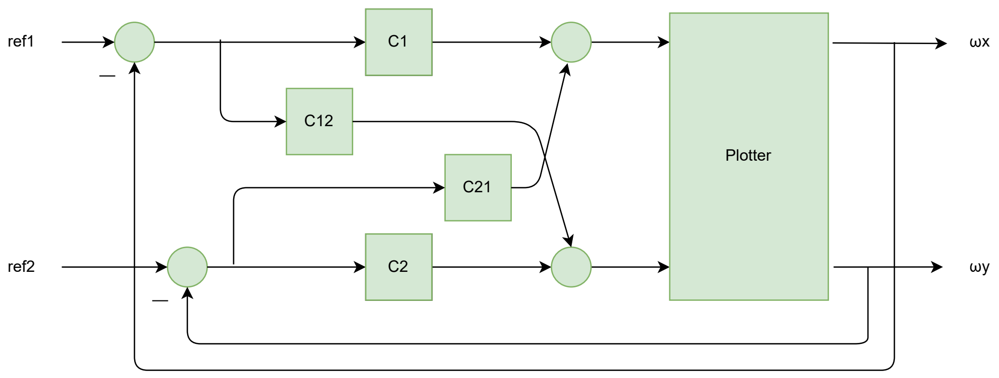


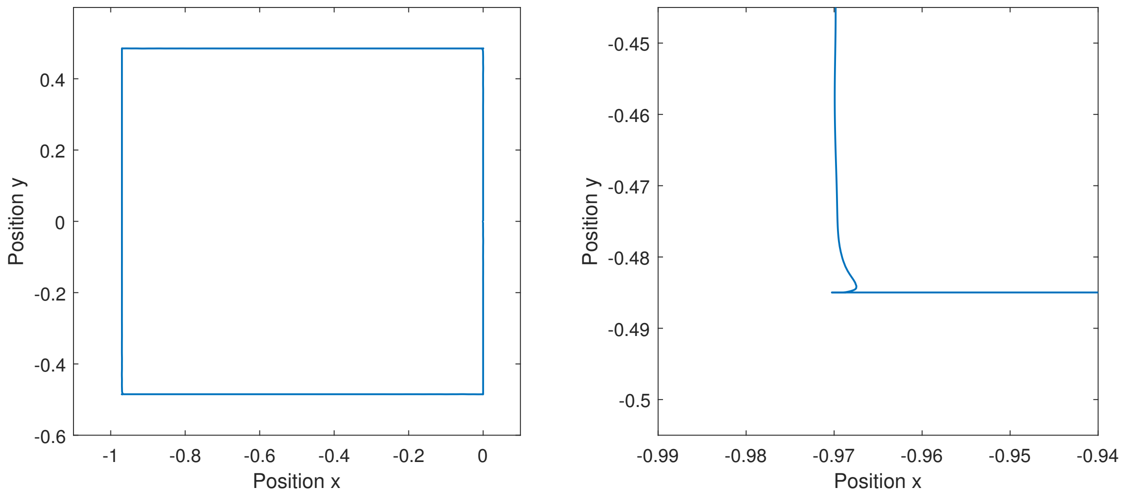
Figure 4.5: $\mathcal{H}_\infty$ fixed structure mimo control setup

That can be done using `hinfstruct` command in Matlab, which is searching for the optimization minimum by varying only the defined parameters. In our case, the parameters were k_p, k_i, k_d and derivative filter parameter in PID controllers $C1$, $C12$, $C21$ and $C2$ (see eq. 4.1). The full generalized plant with $\mathcal{H}_\infty$ synthesis using `hinfstruct` command is implemented in `fixed_stucture.m`. However, for my next experiments, I used a simplified variant of the plant, using only Wt and Wu signals, implemented in `fixed_structure_simple` file.

$$C = K_p + K_i \frac{1}{s} + K_d \frac{s}{T_f s + 1} \tag{4.1}$$

Before I could proceed, I needed to obtain a linear model of the plotter. That was done by trimming the model around zero and using Model Linearizer app in the Simulink. One modification I have done is to delete the integrated outputs ($\theta_{1,2}$ and $pos_{x,y}$) from the model before linearization, so the solver does not have a hard time with them. The results are stored in the `linearization` folder.

Finally, let's see what result this controller can achieve. Reference is the same as usual, but this time derived ω instead of θ, because we deleted these integrated outputs. The drawing from this controller on 4.7.



(a) Drawing **(b)** Detail

Figure 4.7: Square drawing with $\mathcal{H}_\infty$ optimal controller

It looks great, at a glance almost perfect. In the detail of the corner, we can still see an imperfection, which could improve. On the peaks, the control action of this controller overshoots the specified limit, however I included a saturation block, so stays within the specified limit, see fig. 4.8.

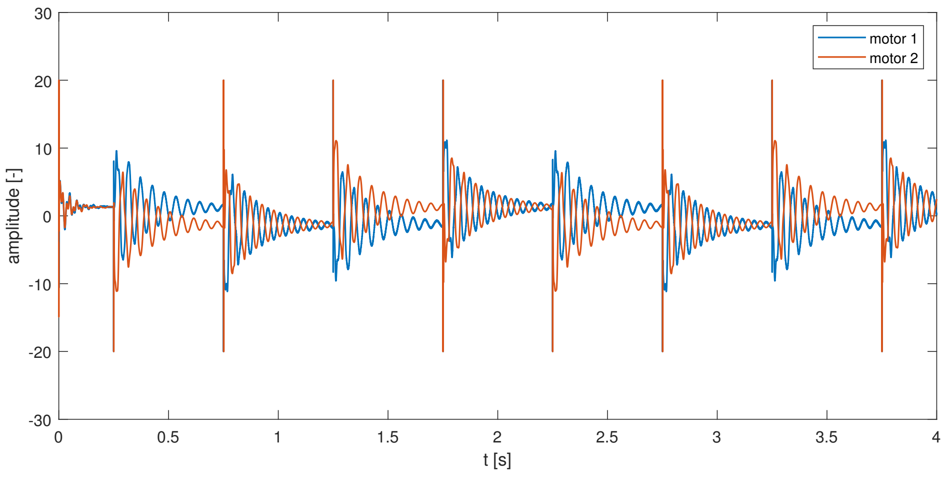


Figure 4.8: Control action of $\mathcal{H}_\infty$ sythetized controller

Let's try to augment this system with an ILC and see, it if gets better. I chose to try the ILC only on $C1$ and $C2$, see fig. 4.9.

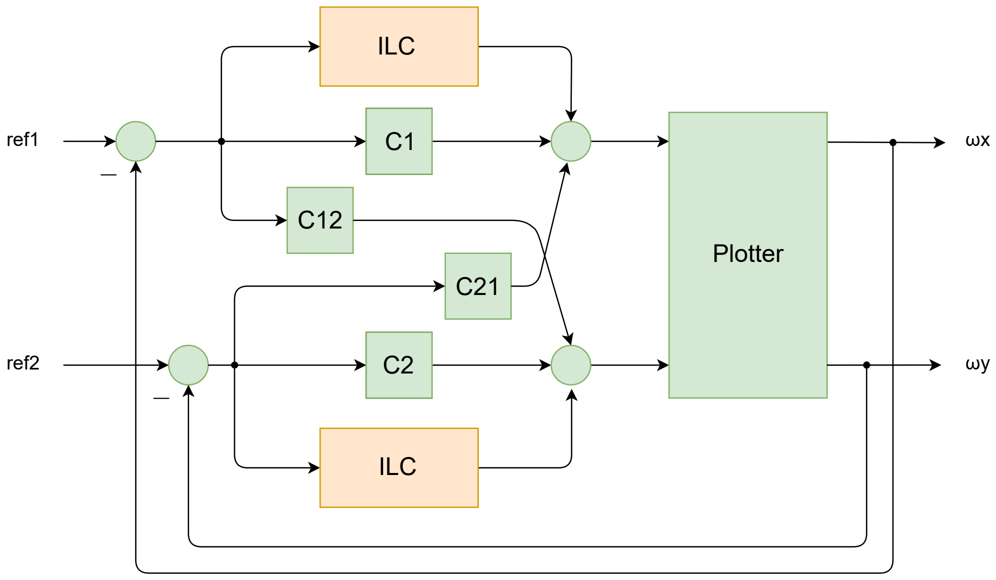
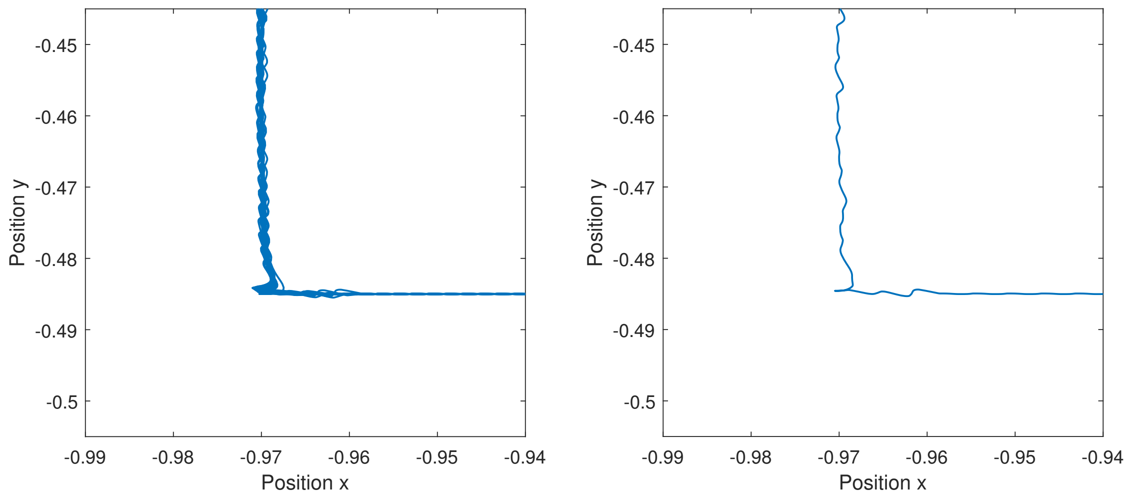


Figure 4.9: ILC mimo control setup



(a) Drawing detail **(b)** Drawing detail; 25th iteration

Figure 4.10: Square drawing - $\mathcal{H}_\infty$ optimal controller with ILC

The results are not great. At this stage we can only assume the problem is the cross-coupling, which this simple design of ILC cannot handle. I tried several variants of filter Q, learning coefficients, but it did not help, so the figures are for consistency and comparison with the very same coefficients as before. Namely $ILC_{kp} = 0.1$ and $0/20$ Hz filter. This problem requires further research and probably a different, more complex solution. For the relevant literature, there exist an older, but very interesting article of $\mathcal{H}_\infty$ synthesis of ILC controller itself [3]. Different techniques with interesting results can be found at [10].

5 | Real Plotter Results

5.1 A Brief Guide of Getting the Plotter to Work

Although I recommend the user better refer to a Gitlab for putting the plotter into operation, let me briefly sum up the steps needed also here. I am assuming basic configuration of Odrive, such as motor specification, encoder type, etc. (see 2.4.1).

The Odrive can accept a wide range of supply voltage, I have used 30 volt 5 ampere power supply. After powering up the Odrive, it first searches for the encoder index on both motors. Thus, before this, the printing head should not be at the end of either axes. Then the user has to manually move printing head to home and calibrate offsets, using `calibration.m` file. Now the plotter is ready to draw.

For generating reference and other parameters, running a configurable `real_plotter_parameters.m` script is recommended. This script can generate triangle, square, star and zigzag patterns with tunable speed and size. Other parameters are sampling frequency (in my experiments it is always 200 Hz) and behaviour after experiment. It also generates other signals and constants needed for the control algorithm to work.

5.2 Experiments

The motors used on the plotter are so powerful that it limits the control challenge. For all the experiments, I did constrain the motors for 4 A peak power. That effectively simulates a smaller and weaker motor. The control setup was serial (fig. 3.2), with the position loop running on Odrive. In some cases, I even lowered the coefficients k_p and k_I of Odrive integrated PID controller, but that will be explicitly stated. It simulates a scenario, where we do not have access to the controller tuning (eg. actuator with integrated control board). Other parameters are auto-generated with the `real_plotter_parameters.m`.

Parameters of ILC are fixed on all experiments. $ILC_{kp} = 0.3$, $ILC_{kd} = 0$, Q filter a simple Moving average with a window of 4. I have briefly experimented with other values, and these I found it to be the best to demonstrate all the important phenomenons. For example from my rough observation a $ILC_{kp} = 0.3$ is quite aggressive, but it only makes the ILC learn faster. It does not have an impact on stability. With introduction of ILC_{kd}, it makes the transient worse. As Q, I did not have time to experiment much, but a “proper” $0/20\ Hz$ low pass filter did not seem to work well.

5.2.1 Drawing Interpretation

There is not always a suitable space for the legend and attaching the same legend to each graph seems redundant. Because of that, I will explain it here. In the figures of experiment plots, there are three signals. Reference, optional disturbance in y axis (see 2.1.2) and drawing itself. Because the repeating drawings often overlap, the improvement can be unclear, which is why I have introduced a “temperature” graph – the newer the line, the lighter the colour. See fig. 5.1. Also, please note that the presented plots depict an estimated position position of the drawing head using a simple kinematics (eq. 2.1), neglecting the belt flexibility.

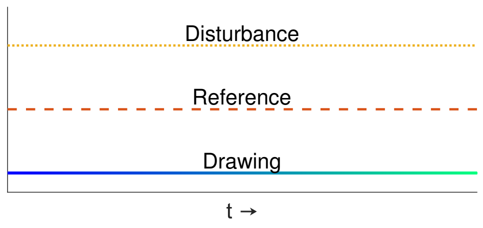
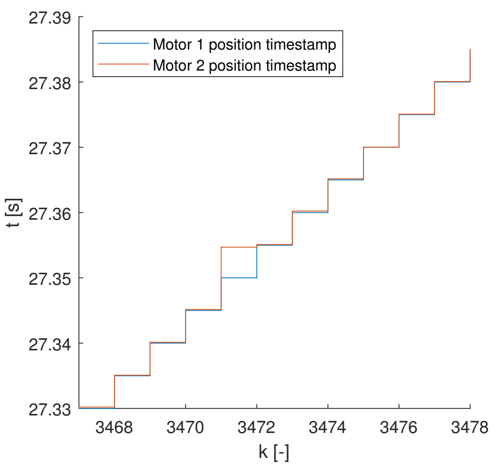


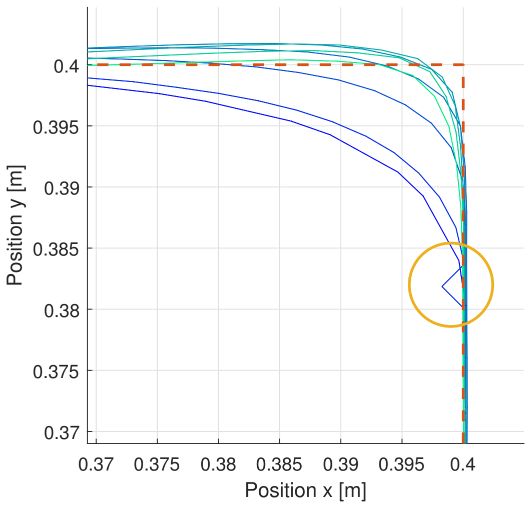
Figure 5.1: Legend

5.2.2 Timing Issues

Let’s perform a simple experiment, square drawing with ILC enabled and observe both the drawing and timestamps of can messages with a position in detail (fig. 5.2). We can see, that sometimes position of one axis is received one step later, then the other, resulting in a small imperfection in a diagonal direction. In my setup current, velocity, and position feedback loops run on Odrive. Simulink handles only the ILC part, and thus the error introduced with the timing and synchronization issues is not critical. Note that I could easily fix it for the thesis graphs, because of the timestamps associated with the position messages. However, I rather decided to attempt to synchronize it in the driver, with no success. The “right” way would be to compile the whole Simulink scheme to a target with a realtime kernel. That way we eliminate the latency from USB too.



(a) Timing of can messages

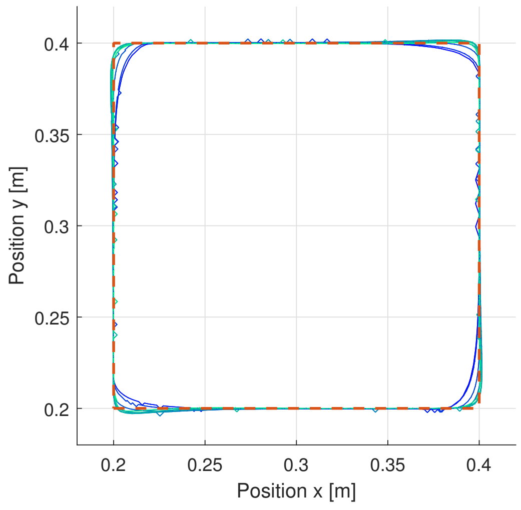


(b) Consequent drawing

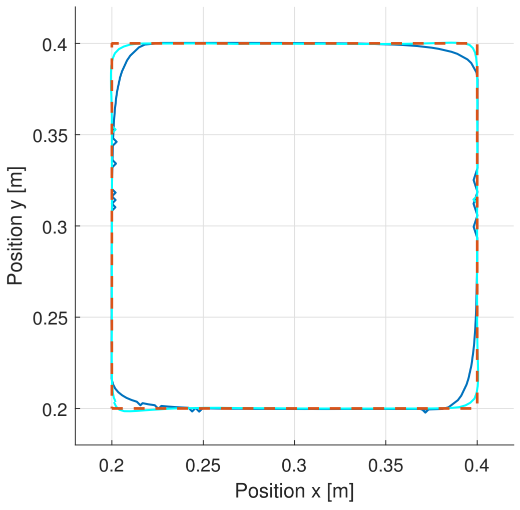
Figure 5.2: Timing of can messages impact

5.2.3 Experiment 1 - Square

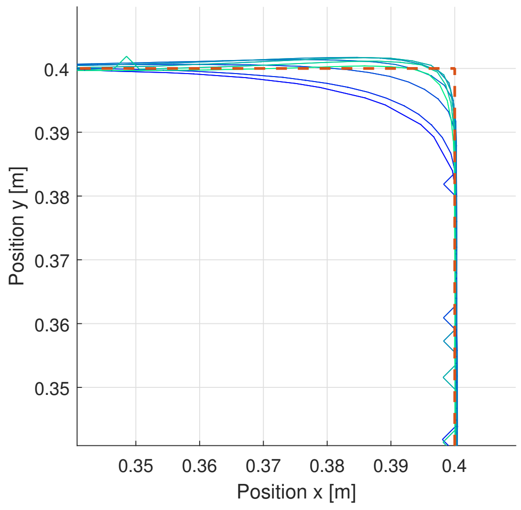
The first plot is a square reference, length of one iteration $t_j = 2\,\mathrm{s}$, final plot is 8^{th} iteration (fig. 5.4). No tuning of Odrive position regulator coeficients was done, they stay on default $-k_p(position) = 20$, $k_p(velocity) = 0.16$, $k_I(velocity) = 0.32$. The control structure is serial (fig. 3.2, Simulink scheme in Apendices A).



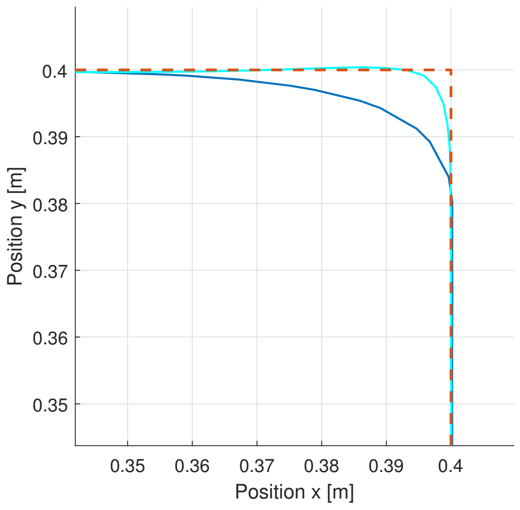
(a) Whole drawing



(b) First and last plot



(c) Whole drawing - detail

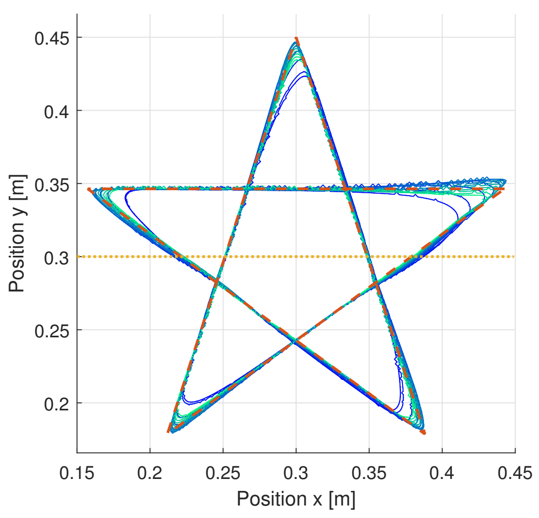


(d) First and last plot - detail

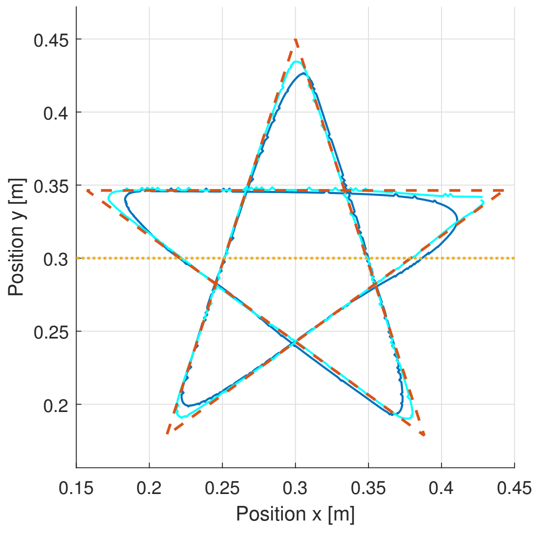
Figure 5.3: Square plots with ILC

5.2.4 Experiment 2 - Star with Disturbance

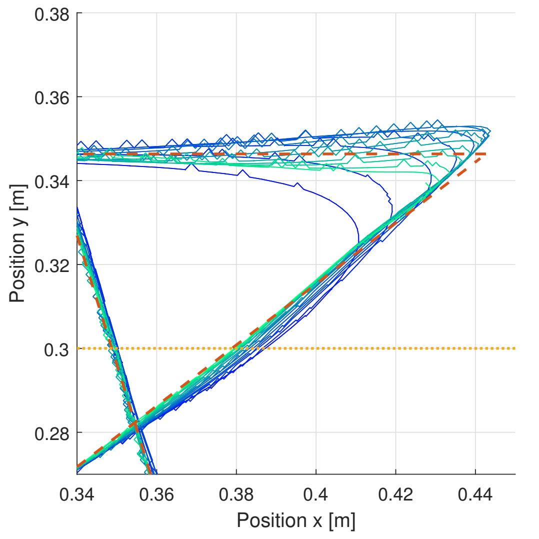
The second experiment (fig. 5.4) is a star reference with disturbance at $y = 0.3\,\text{m}$. Length of one iteration $t_j = 4\,\text{s}$, final plot is 12^{th} iteration. Odrive controller was worsen, all the coeficients halved – $k_p(position) = 10$, $k_p(velocity) = 0.08$, $k_I(velocity) = 0.16$. The control structure stays serial (fig. 3.2, Simulink scheme in Apendices A).



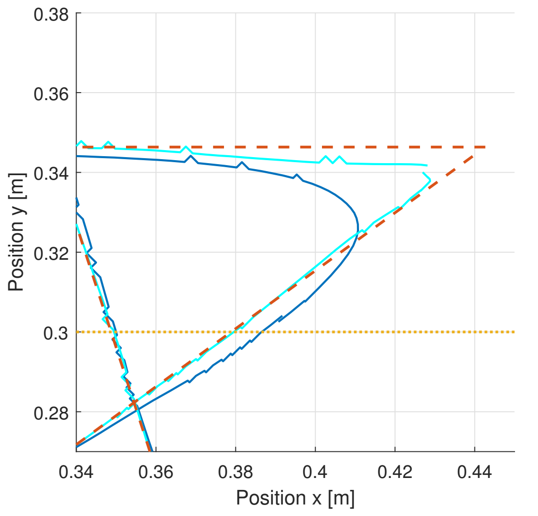
(a) Whole drawing



(b) First and last plot



(c) Whole drawing - detail

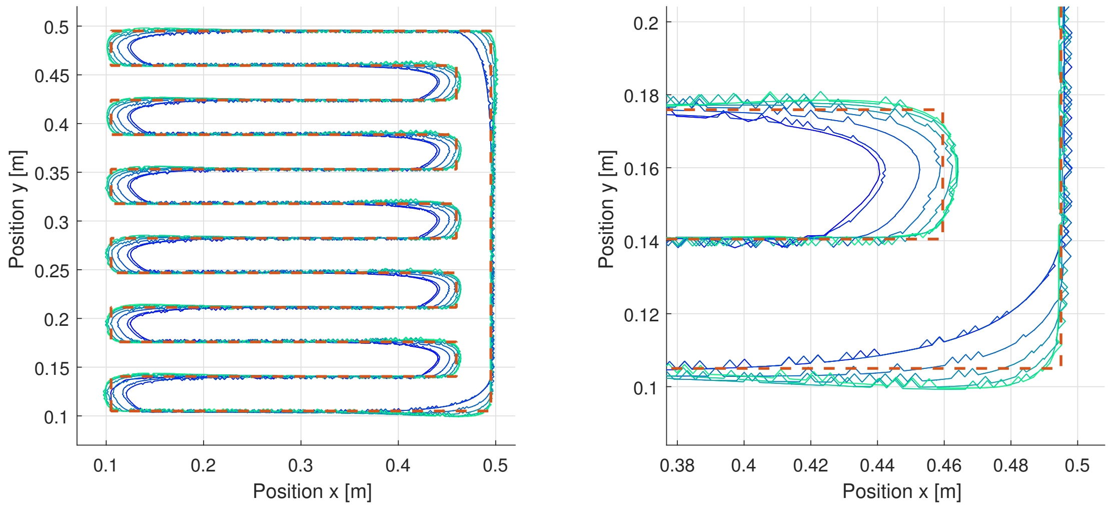


(d) First and last plot - detail

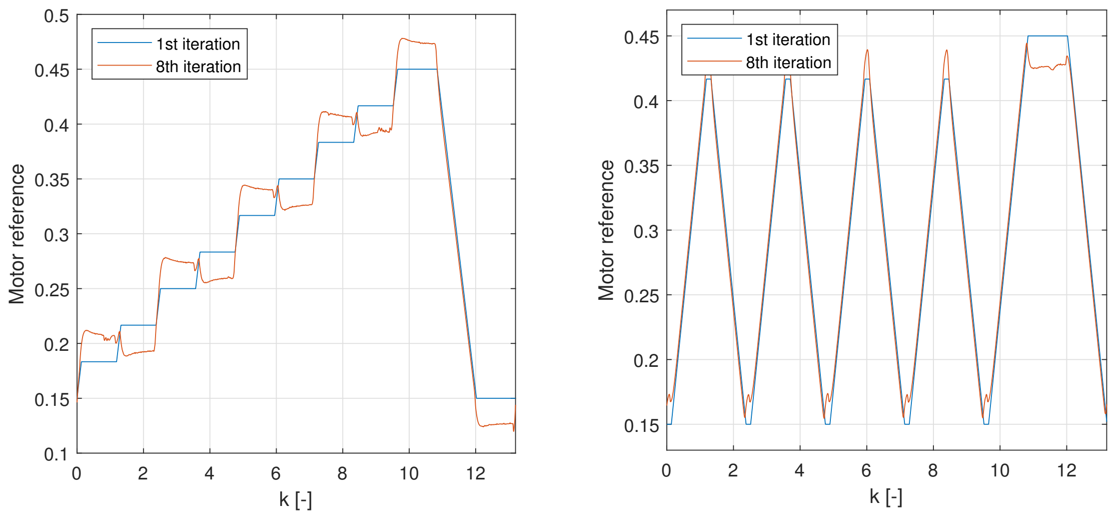
Figure 5.4: Star plots with ILC and disturbance

5.2.5 Experiment 3 - Zigzag Pattern

The third experiment is a zigzag plot. This was inspired by Atomic force microscopy (AFM), where a similar pattern is performed in nanoscale to scan the whole sample. Length of one iteration $t_j = 13\,\mathrm{s}$, final plot is 8^{th} iteration. Odrive controller stays worsen, all the coeficients halved – $k_p(position) = 10$, $k_p(velocity) = 0.08$, $k_I(velocity) = 0.16$. The control structure is serial (fig. 3.2, Simulink scheme in Apendices A). Because ILC in this scenario acts as an input shaper, for demonstration the modified references can be seen in fig. 5.5c, 5.5d.



(a) Whole drawing **(b)** Detail

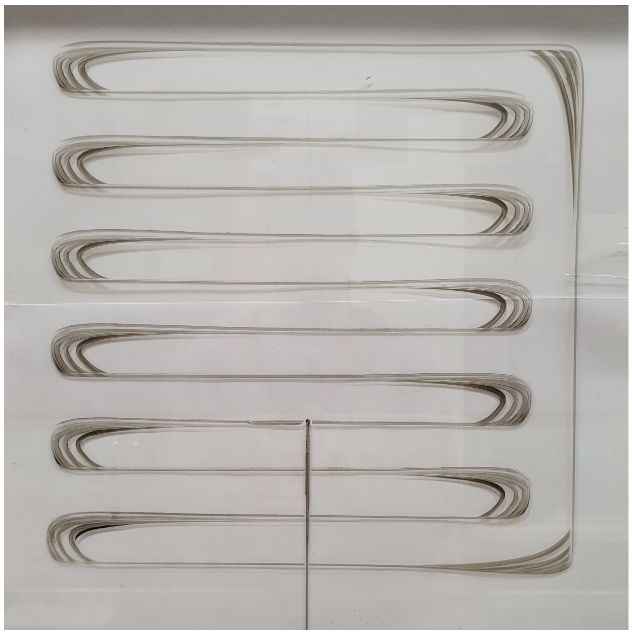


(c) Position reference for Motor 1 **(d)** First and last plot - detail

Figure 5.5: Position reference for Motor 1

5.2.6 Real Drawings

For demonstration, in fig. 5.6 there are real plots on plexiglass with an erasable marker. That can be an attractive demonstration even for the general public. Video of the plotter drawing square and a star is both in the attachment of this thesis and also uploaded on Youtube[1].


(a) Zig zag pattern

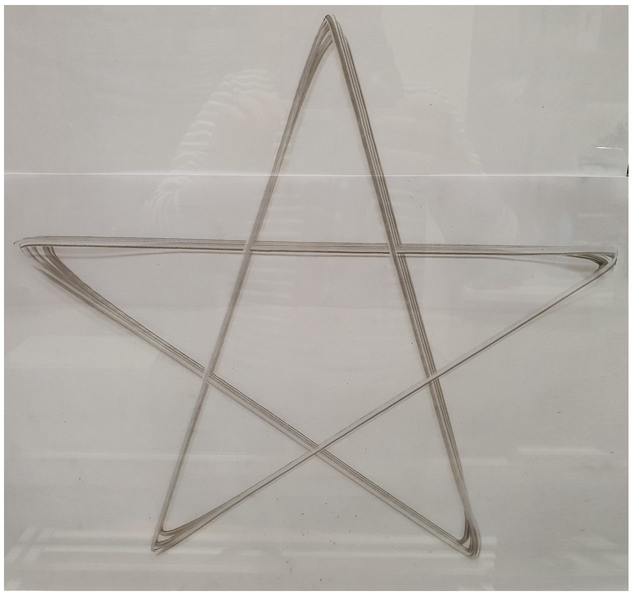

(b) Star pattern

Figure 5.6: Real plots on a plexi glass

[1]https://youtu.be/gUsnRobCMTs

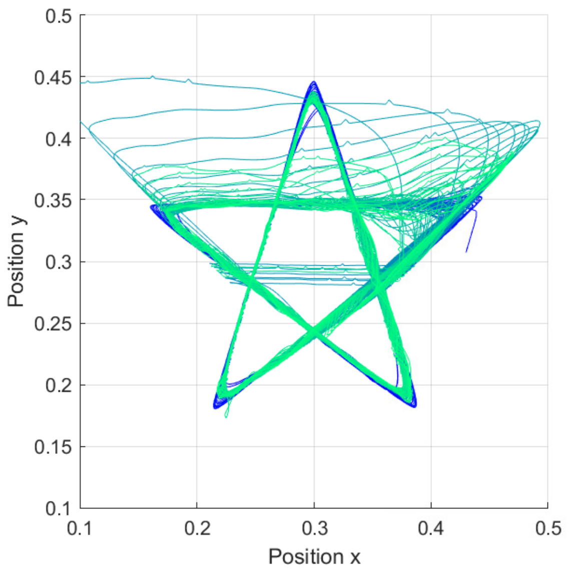


Figure 5.7: divergence of ILC, star reference

5.3 Longterm Instabilities

So far, only the first few iterations of plots have been shown. The behaviour afterwards however shows a divergence (see fig. 5.7, 5.8). What we observe, is a common behaviour of repetitive and iterative learning control. “It is a very common phenomenon to see a learning control law produce substantial decreases in the error in a few repetitions, and then appear to diverge” [5, section 6.4]. Learning transients can be very long, making the system look unstable even for thousand of iterations. In engineering practice, that can be a real problem, because we need not only a convergence, but also good transient behaviour. The cited article shows an extensive theory of the transients. One method the article shows is to bypass the long-term instabilities by using the commands that produce the best performance and disabling learning afterwards (e.g. fig. 5.8, 10^{th} iteration around $t = 20s$). Other, more advanced methods generally rely on fine tuning the Q learning filter accordingly for the system.

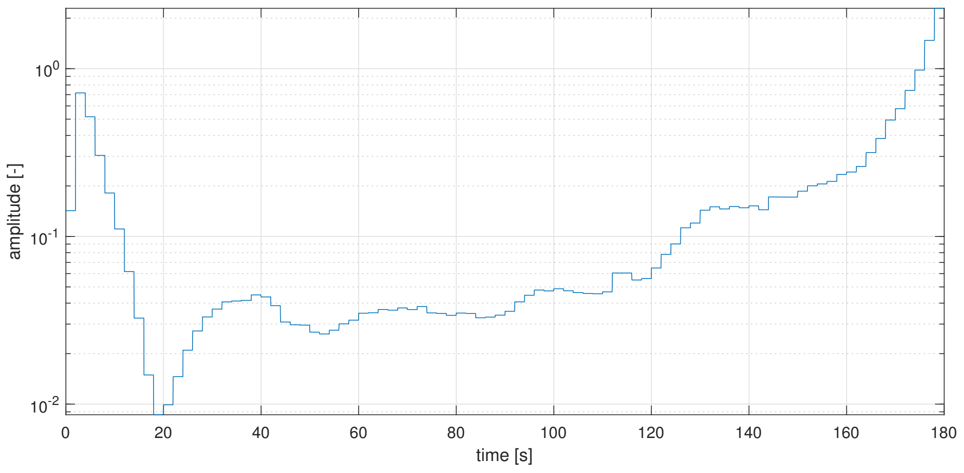


Figure 5.8: Long term performance of ILC (squared tracking error of star drawing)

6 | Conclusion

The first task in my thesis was to design and realize electronic modules for the simultaneously developed mechatronic device – a planar plotter. I dedicated Chapter 2 to this task. Due to some problems in development cycle, I had to design two implementations of a BLDC motor driver to the Simulink interface. Both drivers are opensource, one being Ben Katz's design, the other Odrive while I end up using the latter. In the end, it has been successfully tested, with only some problems arising from the architecture used. The ILC algorithm runs on a PC, which has not a real-time kernel and uses a CAN-USB interface card, which both add latency and jitter. Unfortunately this problem was identified too late in the project, so it was not solved. As a side step, I modified some parts of the Plotter to include an optional disturbance in form of additional friction, which works well.

In the Control Software chapter (3), I have expanded the control challenge. For verifying the model kinematics and getting fell to its dynamics, I performed simple plots. I designed and implemented a full ILC framework and successfully verified the ILC function on an elementary dynamic system. The main parts of this framework are *ILC reference source*, which provides all the control signals for *ILC* controller itself. Simple *Performance evaluator* is implemented, which evaluates the performance of a whole iteration.

At the time when chapter Experiments With the Model (4) was written, unfortunately, model coefficients were only roughly estimated from other sources. Maybe that could be the cause of a fail of implementation of ILC, which I did not manage to get to converge. First I tried a simple SISO-like PID control and then shifted to the MIMO approach, thinking the crosscoupling of the model is the issue. This approach failed, so I tried more advanced techniques. I tried to overcome this coupling by a design MIMO PID controller with $\mathcal{H}_\infty$ synthesis, which generated really nice results alone. The next step was to try the ILC on this well-tuned controller, but I still did not manage to improve the response, actually, it got worse.

In the Real Plotter Results chapter (5) I managed to successfully show off the ILC control. I tried only the basic setup of ILC, without extensive parameter tuning, either ILC or coefficients of other controllers. I tested different shapes and speeds of multiple references, some of them with a disturbance. The repetitive disturbance suppression appeared to be a strong point of Iterative Learning Control.

6.1 Future Work

The hardware part of this thesis is quite complete, with some exceptions. The timing of can messages issue could be solved either in high-level control system (ILC), or on a driver. For that eg. the BeagleBone Blue could be used, where we have much more control over the system resources and no lag by the USB. For that, we need to compile and deploy our Simulink scheme though, which Vehicle Network Toolbox blocks do not support. That should not be a big problem though, because there exist several linux packages which support can communication with decoding using DBC. And this file we already have. The other possible convenience feature is automatic homing, which is done manually so far. Endstops cannot be added without Odrive firmware modifications, but the plotter could home using current measurement.

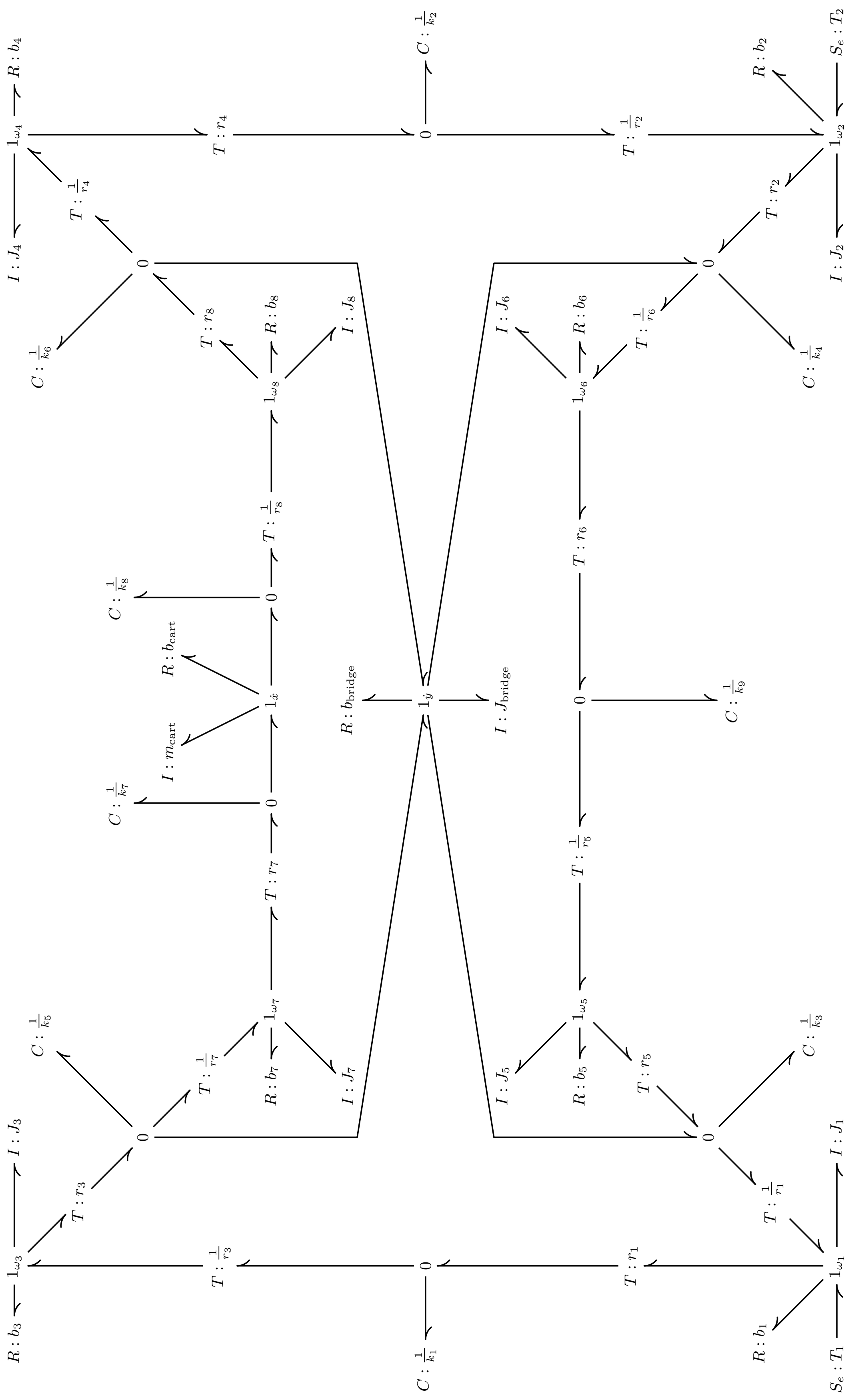
For the model experiments, an obvious way forward is to identify the Plotter and adjust model parameters. Then experiments with it could yield some useful results. Other architectures of ILC can be also interesting. Some need access to a control action of a feedback controller, which our current setup does not support – that is handled by Odrive internally. What Odrive supports, on the other hand, is a current feedforward (Chapter 2.4). In my experiments, I had only access to the position control error from which the path to torque/current is via two derivatives. Maybe an estimator would help... the possibilities of continuing this work are many.

Appendices

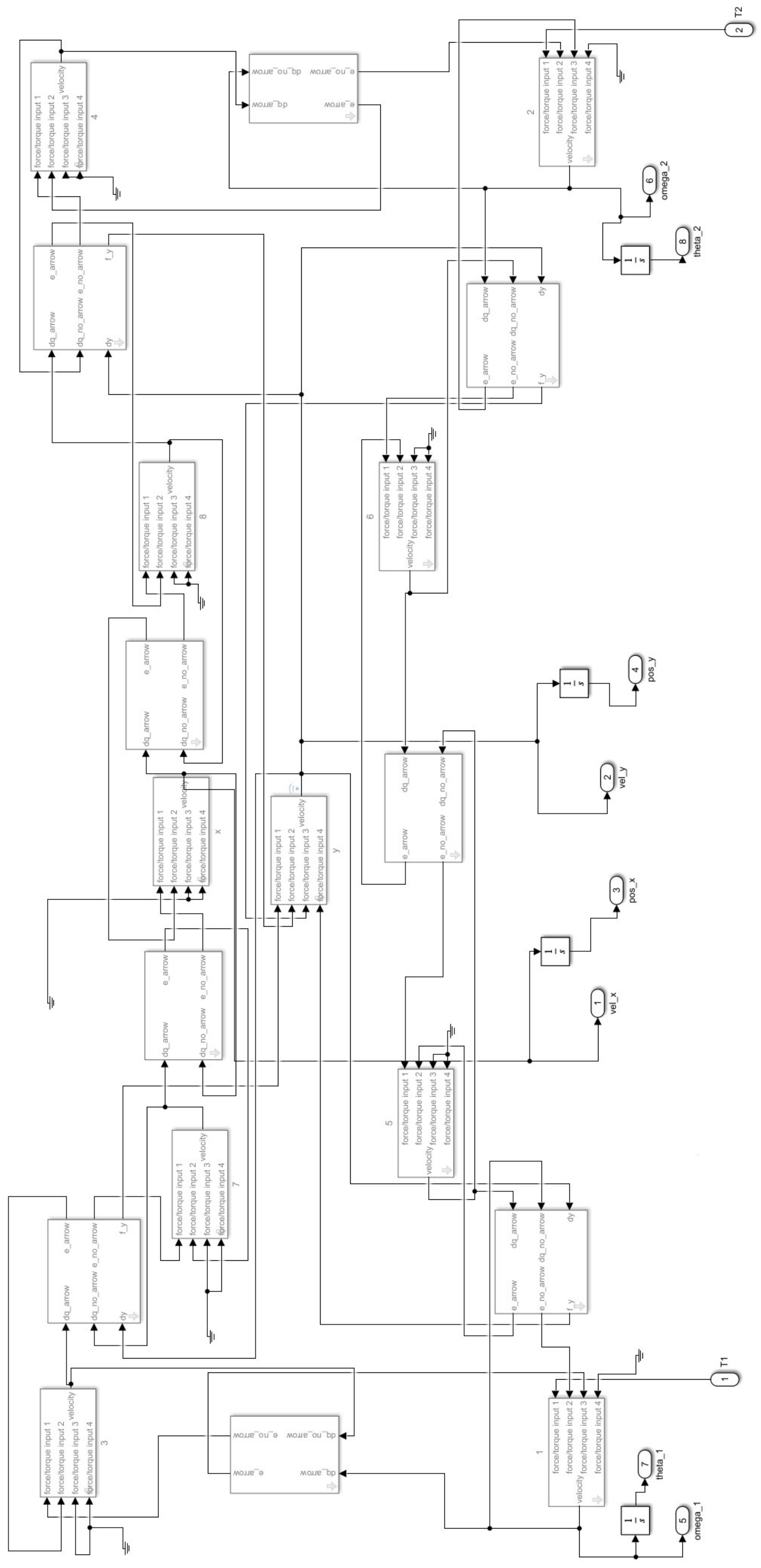
A | Model in Detail

On the next page there is a bond graph that was used to create the plotter model. For completeness, the simulink model by itself is included aswell - but for the detail the reader should really better study the actual Simulink model downloaded from gitlab repository. Author of both the graph and model is Ing. Adam Kollarčík.

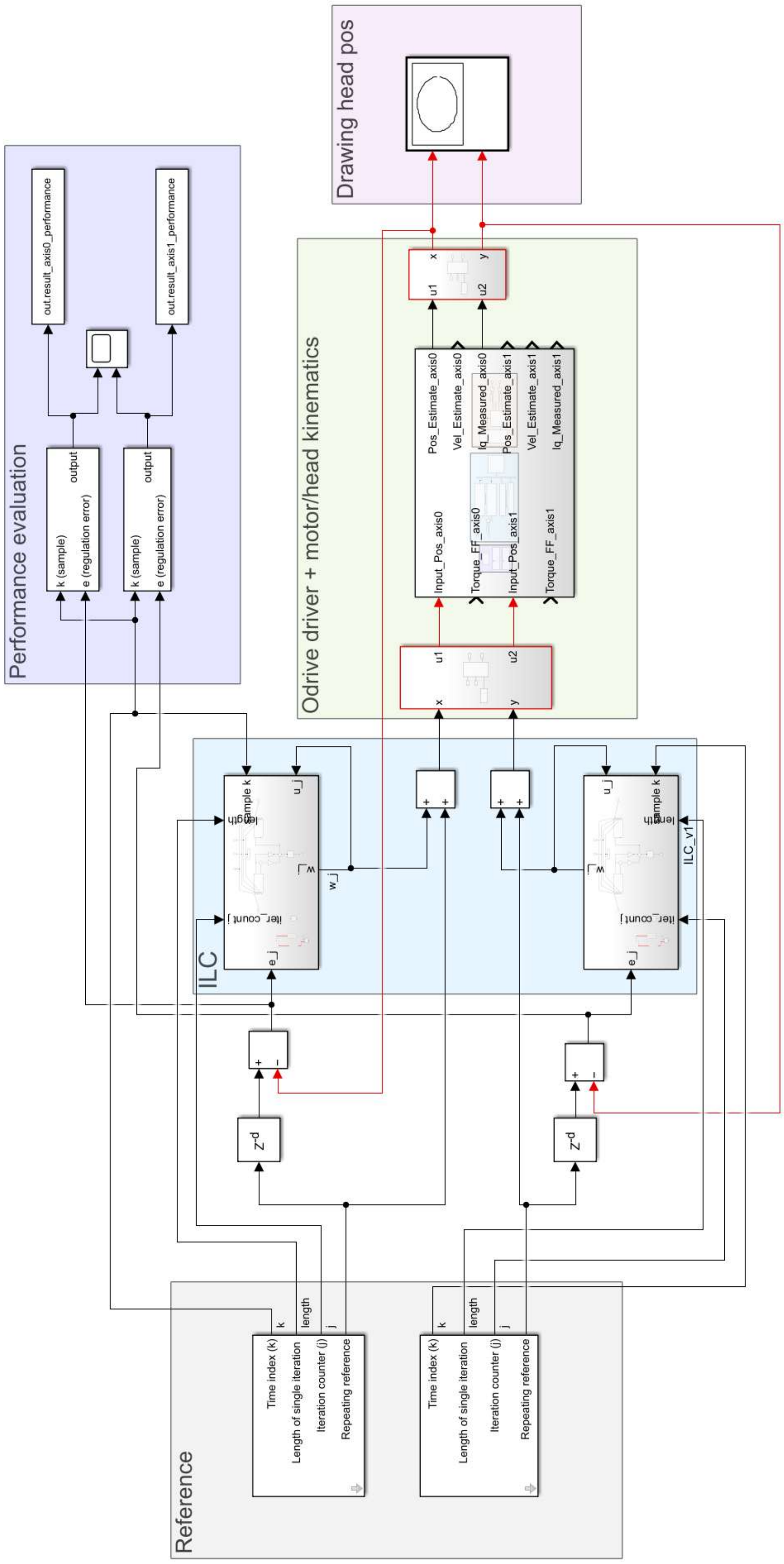
A.1 Bond Graph



A.2 Simulink Model



A.3 Real Plotter Experiments Setup



References

[1] L. Blanken, J. Willems, S. Koekebakker, and T. Oomen. Design techniques for multivariable ilc: Application to an industrial flatbed printer. *IFAC-PapersOnLine*, 49(21):213–221, 2016.

[2] D. A. Bristow, M. Tharayil, and A. G. Alleyne. A survey of iterative learning control. *IEEE Control Systems Magazine*, 26(3):96–114, June 2006.

[3] C. J. Goh and W. Y. Yan. An H infinity Synthesis of Robust Current Error Feedback Learning Control. *Journal of Dynamic Systems, Measurement, and Control*, 118(2):341–346, 06 1996.

[4] D. Kim, J. Di Carlo, B. Katz, G. Bledt, and S. Kim. Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control. September 2019.

[5] Richard W. Longman. Iterative learning control and repetitive control for engineering practice. *International Journal of Control*, 73(10):930–954, 2000.

[6] K. L. Moore, M. Dahleh, and S. P. Bhattacharyya. Iterative learning control: A survey and new results. *Journal of Robotic Systems*, 9(5):563–594, July 1992.

[7] T. Oomen. Learning in machines. *Mikroniek*, (6):5–11, 2018.

[8] S. Skogestad and I. Postlethwaite. *Multivariable Feedback Control: Analysis and Design.* John Wiley & Sons, Inc., 2005.

[9] K. S. Sollmann. Modeling, simulation and control of a belt driven, parallel h-frame type two axes positioning system. Master's thesis, University of Rhode Island, 2007.

[10] J. van Zundert, J. Bolder, and T. Oomen. Optimality and flexibility in iterative learning control for varying tasks. *Automatica*, 67:295–302, May 2016.

[11] Y. Wang, F. Gao, and F. J. Doyle III. Survey on iterative learning control, repetitive control, and run-to-run control. *Journal of Process Control*, 19(10):1589–1600, December 2009.