University of Twente

EEMCS / Electrical Engineering
Control Engineering

Design and testing of embedded control software for the ViewCorrect Plotter

Cornelis Kooistra

MSc Report

Supervisors:
prof.dr.ir. J. van Amerongen
dr.ir. J.F. Broenink
ir. M.A. Groothuis

June 2007

Report nr. 017CE2007
Control Engineering
EE-Math-CS
University of Twente
P.O.Box 217
7500 AE Enschede
The Netherlands

Summary

For the ViewCorrect project, an x-y plotter is built. This is a research setup for testing concurrent real-time software. The context of the ViewCorrect project is to bring different disciplines, involved in a mechatronic system project, in a structured way together and therefore making the traditional gap between them smaller.

One way of bringing different disciplines together is co-simulation. This project has researched the possibilities to co-simulate the CT-based plotter software written in gCSP with the 20-sim model that describes the behaviour of the ViewCorrect Plotter setup.

The existing model of the ViewCorrect Plotter setup has been adjusted and validated. The difference between simulated and measured model is less than five percent. Next a 3D animation model and controllers have been designed. The controllers are capable of controlling the position of the pen.

The plotter software has been made in gCSP. A workflow is presented which allows the user to make a drawing in a CAD drawing package and plot this drawing with the plotter. The controllers, designed in 20-Sim, and the safety layer are embedded in the plotter software.

Co-simulation has been analysed in the scope of heterogeneous system design. A flexible framework is designed to handle the requirements for and challenges related to co-simulation. This framework has been evaluated in a case study. In this case study, the plotter software is tested together with the model that describes the behaviour of the plant and the I/O of the ViewCorrect Plotter setup. Co-simulation is a powerful tool for verification in a model-driven design approach for embedded control systems, which brings engineers from different disciplines in a natural way together. However the success of co-simulation depends on the quality of models used for testing, like the model used for testing the software, and whether the design environments allow for cooperation.

A systematic workflow is presented to isolate and solve causes of unexpected behaviour. It uses parts of an existing approach, but is translated to a workflow for failure analyses at small mechatronic setups. It is valuable to evaluate the workflow for a new research setup during system engineering.

The ViewCorrect Plotter setup has been improved. The pen mechanism is redesigned and a new configuration file is written for the FPGA board, which allows for generating multiple PWM signals at different frequencies to be able to control the pen height. Linear encoders have been implemented to the setup and a PCB is designed to process the plotter I/O. The PCB has the same form factor and is stackable with the existing CE H-bridge PCB to a complete electronic circuit to control a DC motor with an I/O board. A demonstration button, an emergency stop button and a brake stop button can now be connected.

Samenvatting

In het ViewCorrect project is een x-y plotter gemaakt. Deze plotter is een onderzoeksopstelling voor o.a. het testen van real-time parallele werkende software. Het doel van het ViewCorrect project is om de verschillende disciplines in een mechatronisch project op een gestructureerde wijze bij elkaar te brengen en daardoor de traditionele afstand tussen de disciplines te verkleinen.

Een manier om verschillende disciplines bij elkaar te brengen is co-simulatie. Dit project heeft de mogelijkheden onderzocht van co-simulatie van de CT-oriënterende plotter software, gemaakt in gCSP, samen met het 20-Sim model die het gedrag van de plotter beschrijft.

Het bestaande model van de ViewCorrect Plotter setup is aangepast en gevalideerd. Het verschil tussen de gesimuleerde en gemeten waarden is minder dan vijf procent. Daarnaast is er een 3D animatie model en zijn er controllers ontworpen. Deze controllers zijn in staat om de positie van de pen te controleren.

De plotter software is in gCSP ontworpen. Deze software stelt de gebruiker in staat om een werkvolgorde te gebruiken die zijn idee voor een tekening, gemaakt in een CAD teken pakket, laat tekenen met de plotter. De controllers, die ontworpen zijn in 20-Sim, en een veiligheidslaag zijn onderdeel van de plotter software.

Co-simulatie is geanalyseerd in het licht van heterogeneous system design. Een flexibel raamwerk is gepresenteerd voor co-simulatie om aan de eisen voldoen en de moeilijkheden van co-simulatie op te lossen. Dit raamwerk is geëvalueerd in een case study. In deze case study is de plotter software getest samen met het model, die het gedrag van de plotter en I/O beschrijft. Co-simulatie is een krachtig instrument voor verificatie in model gestuurde ontwerpmethode voor embedded regelsystemen, die ingenieurs van verschillende disciplines op een natuurlijke manier bijelkaar brengt. Het succes van co-simulatie hangt af van de kwaliteit van de modellen die gebruikt worden bij het testen, zoals het model bij het testen van de software, en in hoeverre ontwerpomgevingen samenwerking toestaan.

Een systematische werkvolgorde is gepresenteed om een oorzaak van ongewenst gedrag te isoleren en te verhelpen. De werkvolgorde maakt voor een deel gebruik van een bestaande methode, maar is aangepast naar een werkvolgorde voor kleine mechatronische opstellingen. De werkvolgorde zou bij een nieuwe opstelling geëvalueerd moeten worden als onderdeel van de system engineering.

De ViewCorrect Plotter opstelling is verbeterd. Het penmechanisme is herontwerpen en een nieuw configuratie bestand is gemaakt voor het FPGA board. Dit configuratie bestand is uitgebreid met de mogelijkheid om meerdere PWM signalen met verschillende frequenties te genereren. De opstelling is uitgebreid met lineaire encoders en een PCB is ontworpen om de plotter I/O te verwerken. Deze PCB is samen en stapelbaar met een bestaande PCB van de CE vakgroep een compleet elektronisch circuit om een DC motor te besturen met een I/O board. Een demonstratieknop, een noodstop en een remknop kunnen nu worden aangebracht op de opstelling.

List of Abbreviations

- AD :Analog to Digital
- CT :Communicating Threads
- CAD :Computer Aided Design
- CE :Control Engineering
- CPU :Central Processing Unit
- CSP :Communicating Sequential Processes
- DA :Digital to Analog
- DC :Direct Current
- DDS :Data-Distribution Service
- DDL :Dynamic Link Library
- DMPL :Digital Microprocessor Plotter Language
- ECS :Embedded Control System
- EDA :Electronic Design Automation
- FMECA:Failure Mode, Effects and Criticality Analysis
- FPGA :Field Programmable Gate Array
- gCSP :graphical CSP
- GUI :Graphical User Interface
- HILS :Hardware In the Loop Simulation
- HP-GL :Hewlett Packard Graphic Language
- HP-PCL:Hewlett Packard Printed Command Language
- HP-RTL:Hewlett Packard Raster Transfer Language
- I/O :Input/Output
- MCB :Motor Control Block
- OS :Operating System
- OSI :Open Systems Interconnection
- PCI :Peripheral Component Interconnect
- PC :Personal Computer
- PCB :Printed Circuit Board
- PWM :Pulse Width Modulation
- SILS :Software In the Loop Simulation
- TCP :Transmission Control Protocol
- UDP :User Datagram Protocol

Preface

I am happy to present this master's thesis, which concludes my study in Electrical Engineering and will also mean the end of being a student.

In 1994 I started pre-university college in Drachten and after finishing this in 2000, I moved to the HTS in Leeuwarden. Here I learned the basics of electronics, but I realized that I wanted to know more details of the theoretical side. This made me move to the University of Twente. I arrived at Kampf'47, a student flat near the University.

Kampf'47, the staff and students of the University introduced me to academic life. I will look back with great feeling on the celebrations by tradition like 'feutenfust' or Christmas dinner, our soccer trips to Germany and the countless times we were debating about politics, economics, life and of course soccer.

I would like to thank to following people in random order. My supervisors Job van Amerongen, Jan Broenink and Marcel Groothuis for providing me this assignment and their supervision and support. The technical staff, Marcel Schwirtz, Gerben te Riet o/g Scholten and Alfred de Vries for their help with constructing the ViewCorrect Plotter setup. Also, I would like to thank Pieter Maljaars and Peter Visser for their help during our weekly meeting or otherwise. I want to thank the remaining people at the Control Engineering department, under supervision of Job van Amerongen, for making it a pleasant time to work here and providing an excellent research environment.

Finally, I would like to thank my family, family in-laws and my girlfriend Aly for their continuing support during my study.

Cornelis Kooistra
Enschede, June 2007

Contents

1 Introduction

1.1 The ViewCorrect Plotter

The ViewCorrect Plotter is a research setup for testing concurrent real-time control systems (Kuppeveld and Sprik, 2006). It is an x-y plotter, where both axes are able to move independently of each other. The advantage of the plotter is its simplicity in construction and the possibility to operate at high velocities. In this way, time constraints and accuracy of real-time software can be verified. It is also possible to use the plotter as a demonstration setup for embedded systems or as an experimental setup for advanced controllers. In Chapter 2, the plotter is discussed in more detail.

1.2 Research context of the project

At the Control Engineering (CE) group, one of the research directions is embedded control system design. A research topic is distributed control systems. A part of this work is done in the context of the following PhD project: *Predictable co-design for distributed embedded control systems* (Groothuis et al., 2006).

The purpose of this research is to provide methodological support, including (prototype) tools, for the predictable design of distributed hard real-time embedded control systems for mechatronic products.

This methodology makes use of three main components: views, multidisciplinary core models and correctness preserving code generation. These three components will bring the disciplines involved in a mechatronic system project towards each other and make the traditional gap between them smaller. The disciplines use their own preferred tools, but the models in these tools are related to a multidisciplinary core model. In this way it is possible to view the impact of a decision made by another discipline. This methodology aims to relax the tension between design cost and design time on the one hand and quality (in particular reliability and robustness) on the other hand. The methodology will be tried out using several test setups: the plotter and the Production Cell setup (van den Berg, 2006).

1.3 Goal of the project

The main goal of this project is to enable to give demonstrations for new design methods for embedded control software with the ViewCorrect Plotter in the context of the ViewCorrect project. The context of the ViewCorrect project is to bring different disciplines in a structured way together. One way of bringing different disciplines together is co-simulation.

This MSc-project will research the possibilities to co-simulate the Communicating Threads (CT)-based plotter software written in graphical CSP (gCSP) (Jovanovic et al., 2004) with 20-Sim (CLP, 2007). In 20-Sim a model that describes the behaviour of the setup and a 3D model of the plotter are made. In this way, it is possible to test the software together with its physical environment without using the real plant. The purpose of this approach is to develop the software in such a way that it will run first time right on its equipment.

1.4 Outline of the report

In Chapter 2, the ViewCorrect Plotter setup is described in more detail. Chapter 3 discusses the model of the setup, the validation of this model, the design of the controllers and a 3D animation model. Chapter 4 deals with the plotter software. It describes a design approach to make a drawing with the plotter and the design of the plotter software in gCSP. In Chapter 5, co-simulation is analysed and a framework is presented to handle the challenges related to co-simulation. This chapter concludes with a case study of co-simulation between gCSP and

20-Sim. Chapter 6 discusses a systematic workflow for failure analyses specialized for small mechatronic setups. The conclusions and recommendations can be found in Chapter 7.

2 ViewCorrect Plotter Setup

In this chapter the ViewCorrect Plotter setup is described extensively. First the specifications and requirements (2.1) are discussed. This is followed by a description of the mechanical (2.2), electrical (2.3), software (2.4) and safety parts (2.5). Figure 2.1 shows the ViewCorrect Plotter.

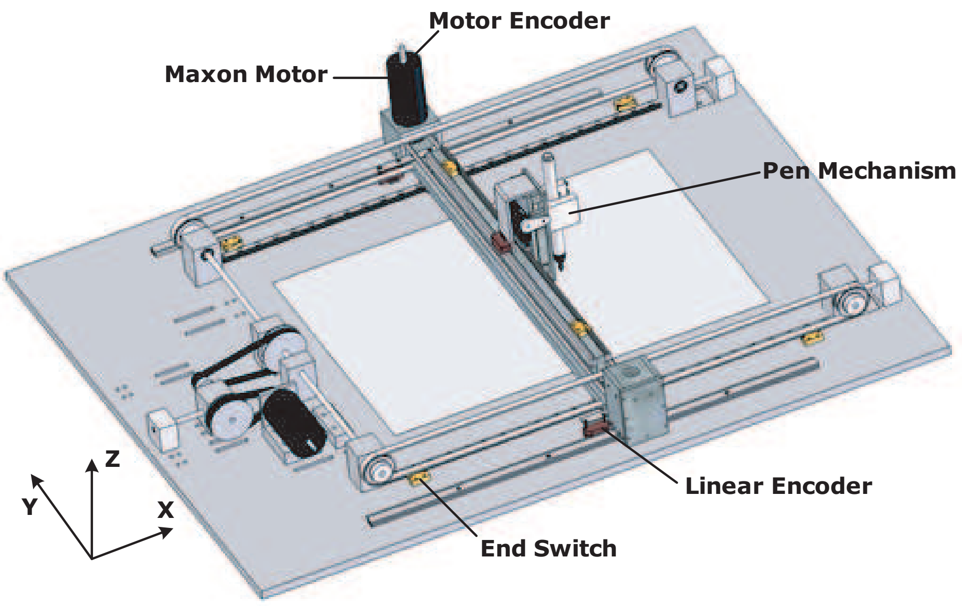


Figure 2.1: ViewCorrect Plotter.

Figure A.5(a) shows a schematic topview of the ViewCorrect Plotter with all the named components.

2.1 Specifications and requirements

This section describes the current specifications and requirements.

2.1.1 Specifications

- Direct Current (DC) Maxon motors are used for the x- and y-axis. Models of these motors are available in the model libraries of 20-Sim. The motors are controlled by Pulse Width Modulation (PWM).
- End switches are used to detect reaching end of rail. If an end switch is reached, then enough time is available to stop the mechanism safely.
- The draw accuracy is set to 0.1 mm when the motion is finished and 1 mm during the motion at maximum speed. However, it is not needed to have a higher accuracy than the pen point width. The previous accuracy was 0.75 mm for the y-axis and 0.4 mm for the x-axis when the motion is finished. The draw accuracy is 0.03 mm during a motion with a maximum velocity of 0.2 m/s and 0.005 mm when the motion is finished for the x-axis. For the y-axis, these parameters are 0.1 and 0,04 mm. This is described in Chapter 3.

- The position measurements is done by motor and linear encoders. The encoders provide feedback for the controllers.
- The drawing area is a A3-size (420 by 297 mm).

2.1.2 Requirements

- The setup should be easy accessible, because of demonstrating purposes. Changing paper should be a simple operation, because this occurs frequently.
- Safety is a important part of a test and demonstration setup. The setup has to be robust and has to designed in such a way that damage by faulty control or human faults are prevented.
- The setup has to be designed to be prepared for distributed control.
- The setup must be equipped with a demonstration button. When this button is activated, a demonstration will start. However this demonstration button is not implemented yet.
- The maximum velocity should be limited to 0.5 m/s. This is due to safety reasons, but this velocity is still fast enough to make a demonstration interesting to see. Due to the current status of the setup, the setup is still open which might be hazardous for bystanders, the maximum velocity is set to 0.2 m/s.
- The size of the plotter and his peripheral equipment should not exceed the size of a lab table.

2.2 Mechanical

Most parts of the plotter are made of aluminum. The axes are made from steel rod. The base plate is made from hardened aluminum. The weight of plotter setup is 18 kg. The rotation of the motor is converted to a translation by toothed belts and pulleys. The guidance system is implemented by rails.

The z-axis is the movement of the pen. It has two positions, on and off the paper. This movement is implement by a servo motor and an up and down mechanism. The x-axis is the longest side. One motor is connected to a steel wire axis which drives both sides. The setup is prepared for driving each side with a single motor. The y-axis is driven in the same way as the x-axis. The pen mechanism moves along a guidance rails back and forth. The mechanical part of the setup had still some shortcomings, which are described in the following subsections.

2.2.1 Improvement of mechanical parts

Some mechanical parts, especially the x-axis pillars with bearing, were not constructed precise enough. These pillars were pulled straight using screws, which harmed the construction more and more by time. Also the movement of the y-axis showed a non-linear behaviour caused by the construction. These shortcomings have been repaired partly in this project.

2.2.2 Redesign of the pen mechanism

The current design of the pen mechanism is new. In the previous project the pen was fastened with tie wraps. The redesign and realization of the pen mechanism is described in Appendix A.1.

2.2.3 Linear encoders

The exact position of the pen can be measured with the linear encoders. Both axes use an linear encoder. The x-axis uses two linear encoders. In this way, it can be verified that the y-axis is moving perpendicular to the x-axis and during operation it will give information about possible friction or position error between the two sides of the x-axis. Especially when each side will be driven by a single motor in future projects. In the previous project, the linear encoders were not implemented. The design and realization of the implementation of the linear encoders is described in Appendix A.3.

2.3 Electrical

To control the setup a Embedded Control System (ECS) is used. The ECS consist of a PC/104 (Seco, 2007) with an Anything I/O board (Mesanet, 2007). The PC/104 with an Anything I/O board is already used in other setups at the CE laboratory. A PC/104 is a Personal Computer (PC), but with a different physical construction. It is based on a stackable circuit board. The Operating System (OS) system on the PC/104 is a Linux kernel with a real-time package. This makes it possible to run software from 20-Sim or gCSP.

The PC/104 is connected by a Peripheral Component Interconnect (PCI) bus with the Anything I/O board. This board has 72 general purpose Input/Output (I/O) ports connected to an Field Programmable Gate Array (FPGA). These I/O are available on three 50 pin connectors. The board is stacked on the PC/104.

At the start of this project electronic hardware from the Production Cell were used. A switchboard (van den Berg, 2006) was used to connect to different signals to the end switches, servo motor and encoders. A motor amplifier, H-Bridge (van den Berg, 2006), is used to drive the Maxon motors. However the switching board was not suitable to be used for the ViewCorrect Plotter setup. The linear encoder and the servo motor can not be connected to this board, it contains unnecessary expensive components and for distributed control the board can be simplified. Therefore a new electronic board and named as Motor Control Block (MCB) is designed in this project. The design and realization of the printed circuit board for the plotter I/O is described in Appendix A.4. Figure 2.2 shows a schematic view of the electronics.

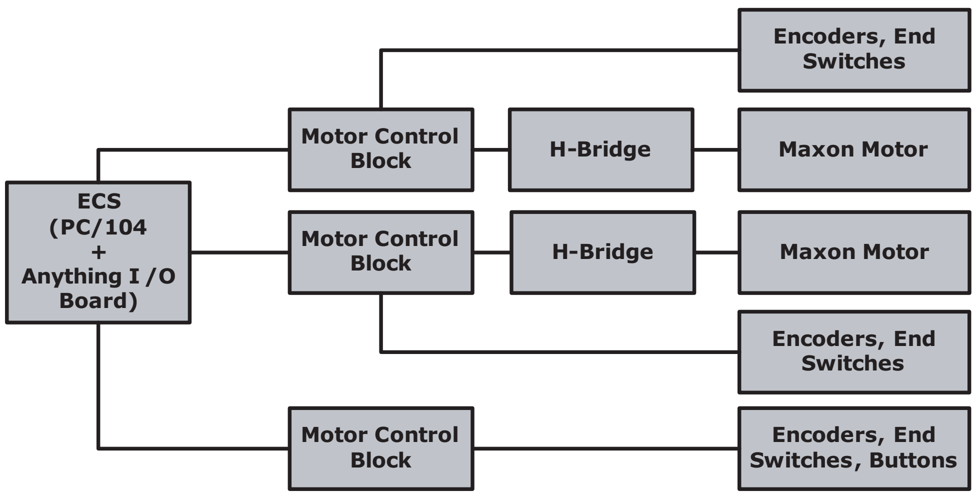


Figure 2.2: Schematic view electronics ViewCorrect Plotter setup.

2.4 Software

Two types of software are written for this setup: real software for the PC/104 and the firmware for the FPGA hardware.

2.4.1 FPGA configuration file

The FPGA at the Anything I/O board has to be configured. The current FPGA configuration is the configuration of (Groothuis, 2004) extended with possibility to generate multiple PWM signals with different frequencies. In the old situation, it was only possible to generate a fixed PWM frequency for all PWM signals at 16.3 kHz. This is implemented in this project because of the redesign of the pen control.

Redesign of the pen control

Another aspect of redesign in this project is the pen control. The pen is driven by a servo motor. The servo motor and thus the position of the pen is controlled by a PWM signal. In the previous design the pulse was generated from the plotter software. In order to generate a pulse with a width resolution of 0.1 ms, a clock frequency of 10 kHz in needed in the plotter software. This is a high and undesired requirement for the plotter software. The redesign and realization of the pen control and thus the current FPGA configuration is discussed in A.2.

2.4.2 Plotter software

The plotter software is needed to use the setup. The previous plotter software was developed with 20-Sim. The movements of the plotter were limited to motion profiles which are not suitable for drawings. The current plotter software is designed in gCSP. This is described in Chapter 4. The CE- and ForSee toolchain is used to send the plotter software to the setup (Buit, 2005) (Posthumus, 2006).

2.5 Safety

In the setup, a couple of safety measures are implemented. In normal operation mechanical parts do not interfere with each other. Due to faulty control software or human faults, it is possible that a moving part is going to collide with other parts. These collisions are prevented by use of end switches. In case an end switch is hit, the moving part is stopped by actively braking the responsible motor.

An extra safety measure will be a emergency stop and a brake stop. An emergency stop is the highest level of safety and disconnects the voltage supply. A brake stop will brake electronically the motors. The difference between both stops is that pressing the emergency stop the moving parts are losing velocity by friction or collision. In case of a brake stop the moving parts are stopped electronically, but the setup is still connected to the power supply. Last two described safety measures are not implemented in the current setup.

2.6 Conclusion

- The pen mechanism has been redesigned and implemented. This mechanism can replace the pen easily and is prepared for future projects with another device like a milling cutter or inkjet head.
- The FPGA configuration file is extended with possibility to generate multiple PWM signals with different frequencies to be able to control the pen height.
- On both axes linear encoders are implemented. For this purpose different options have been analysed on costs, construction time, accuracy and robustness.

- A MCB is designed and realized to process the plotter I/O. The MCB has the same form factor and is stackable with the existing CE H-bridge PCB (van den Berg, 2006) to a complete electronic circuit to control a DC motor with an I/O board. The board is equipped with an additional I/O connector for measuring or extension with additional electronics. The additional connector is interconnected with the connector to the I/O board. Consequently, it is possible to measure the signals at the I/O board during operation.

Due to lack of time some features of the ViewCorrect Plotter setup have not been implemented. At this moment, paper is fixed to the bottom plate. This should be replaced with some sort of clipper.

Besides a demonstration button, an emergency stop button and a brake stop button should be implemented. The demonstration button should be connected to a button connector of the MCB and a software process should detect a change of state and start a demonstration program stored in flash memory of the PC/104. The brake stop should also be connected to a button connector of the MCB. A predefined process in the FPGA configuration file should process this to a brake signal to the H-Bridge. The setup is still open, consequently dust and dirt can harm the construction. Besides it might be hazardous for bystanders, because the plotter can move with high velocity. It is recommended to make some sort of transparent cap.

If higher accuracy is needed it is recommended to redesign part of the gear of the x- and y-axis, because visual inspection shows still some non-linear behaviour. This was also visible in the validation of model that describes the behaviour of the ViewCorrect Plotter setup.

In the next chapter, the model of the ViewCorrect Plotter setup, validation of this model, design of controllers and a 3D animation model are described.

3 Modelling, Validation and Controller Design

3.1 Introduction

Modelling and simulation is used for various parts of the project. The model of the ViewCorrect Plotter setup and its 3D animation extension model are used to design controllers, simulate the effects of changing dynamic parameters and for co-simulation with the plotter software. In case of co-simulation, the model will be used to test the plotter software together with its modelled physical environment.

The previous model of the ViewCorrect Plotter setup (Kuppeveld and Sprik, 2006) has been verified, but has not been validated. This model does also not include all properties of the setup, like the consequences of differences in elasticity in the belts between both x-axes.

This chapter describes the new model (3.2) and its validation (3.3), the design of the controllers (3.4) and a 3D model (3.5).

3.2 Model of the ViewCorrect Plotter setup

The current model is divided into five functional submodels. This division makes it easier to reuse the separate models for controller design or co-simulation. Figure 3.1 shows the top level of the model.

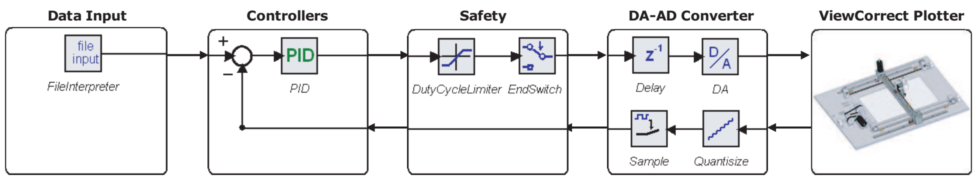


Figure 3.1: Top level model of the ViewCorrect Plotter setup.

Data Input: The data input is a motion profile for the plotter. The motion profile can be a motor signal profile or a motion profile generated by the drawing to motion translator. This profile contains a motion to make a user-designed drawing. The drawing to motion translator is discussed in section 4.4. The data input submodel is connected to the controllers.
Controllers: This submodel contains the two controllers for the x- and y-axis. The design of the controllers is discussed in section 3.4. Another part is the control for the z-axis. This is a fixed value for the up or down position of the pen.
Safety: In this model, two safety measures are included. The first safety measure is a duty cycle limiter, which limits the steering value to a maximum (software safety measure). This safe value corresponds to a duty cycle, where the motor can prevent a collision with the pillars in case an end switch is hit. The second safety measure is the usage of end switches (hardware safety measure). In case an end switch is hit, the motor will brake and the duty cycle becomes zero. The controllers and the safety layer are embedded in the plotter software. This is discussed in section 4.5.
Digital to Analog (DA) and Analog to Digital (AD) conversion: The input for the actuators is converted to an analog signal. The signals from the sensors are converted to the digital domain.
Plant model ViewCorrect Plotter: This submodel contains the dynamic model of the plant. The structure of the plant model is depicted in Figure 3.2. It is split up in *H-Bridge*, *DC-Motor* and *Mechanical*. The results of the validation of motor model concluded that the previous motor model was incorrect with differences up to 200 percent. The model of the motor is replaced with

a bond graph model with parameters from the datasheet. This model contains less details as the model from the 20-Sim motor library. However the missing details, like temperature of the motor, are not needed for co-simulation or controller design. The model of the y-axis has been adapted to include the consequences of the elastic differences in the belts between both x-axes.

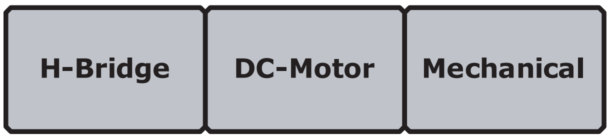


Figure 3.2: Structure plant model ViewCorrect Plotter.

The details and parameters of the model can be found in Appendix B.1. The validation of the model is described in the next section.

3.3 Validation of the ViewCorrect Plotter setup model

The model is validated, because it results into a more correct and accurate model. This is needed for the co-simulation case study, which is described in section 5.5. An incorrect or not accurate enough model can give the idea that the software might be right while this is not the case. The model described in the previous section is validated in two parts.

The first part are the submodels *data input* and *DA and AD conversion* and the motor part of the *plant model*. The motor part of the *plant model* is the H-bridge and the DC-motor. The second part is the complete *plant model*, *data input* and *DA and AD conversion*. In this way the model is validated more accurately.

The motor model produces significant better results as the previous motor model of (Kuppeveld and Sprik, 2006). Now the difference between model and the real setup is at the most 0.5 percent for both axes. The results can be found in Figure B.2.

The complete model, but still without the controllers and safety submodels, is validated after the first part validation using a steep motion profile. The estimated parameters of the friction, caused by bearings and guidance rails, have been changed as a result of the validation results. The results and figures with the simulation and measuring data can be found in Figure B.3. The differences between the simulated and the measured data is below five percent. From this analysis it can be concluded that the model is accurate enough for controller design and (co-)simulation. The next section describes the design of the controllers.

3.4 Controllers design

The motors needs to be controlled in such a manner that the position of the pen follows the lines of a given drawing. The control variable is the angular position of the motor, which represents a certain movement of the pen. The maximum velocity specifications of the motion profile and accuracy of the plotter can be found in section 2.1.

The focus of this project is not on the design of a controller. Therefore controller design starts with a basic controller: a PID controller. The advantage of a PID controller is its simplicity and consequently a low number of computations. Besides a PID controller is able to minimize unpredictable errors due to mechanical vibrations or and a PID controller is quite robust against small differences in plant parameters compared to the plant model. The axes move independently, so two controllers will be designed. The parameters are determined with the Ziegler-Nichols method (Astrom and Wittenmark, 1997).

As shown in the results in Figure B.4, the designed PID controllers can fulfill the specifications.

The complete model as depicted in Figure 3.1 is validated, the results are shown in Figure B.5. The error between simulated and measured encoder pulses is periodical during a motion. It looks like the dynamics are caused by the gear construction, because the number of periods is a multiple of the number of rotations of the motor axes.

3.5 3D animation extension of the model

A 3D animation is made using the 20-Sim animation toolbox. The 3D model is coupled to the dynamic model, but this is not necessary. The animation might be used stand alone, like a real-time visualization moving along with the plotter. The 3D model visualizes the movements of the ViewCorrect Plotter setup. Animating the 3D model with the simulation data from the dynamic model enables a quick insight in the correct functioning of the setup. It can be used for a couple of reasons. First it can be used to verify the working of the model as a whole. Secondly it can be used for demonstrating purposes. Besides it will be used in the visualization of the co-simulation between 20-Sim and gCSP. This is discussed in Chapter 5. Figure 3.3 shows the 3D animation model.

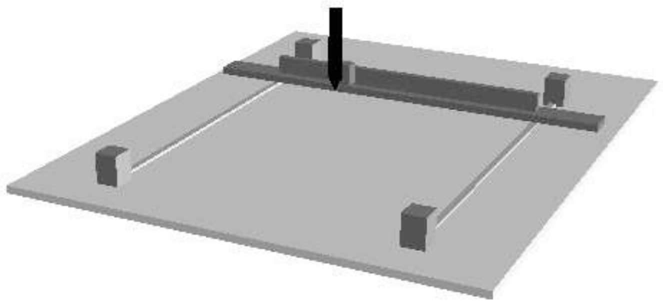


Figure 3.3: 3D animation model of the ViewCorrect Plotter.

3.6 Conclusion

The existing model of the ViewCorrect Plotter setup is adjusted and validated. The difference between simulated and measured parameters is less than a maximum of five percent. The 3D animation model is made for enabling a quick insight to verify the dynamic model as a whole and for demonstrating purposes. Controllers for both axes have been designed according the Ziegler-Nichols method. The accuracy of the model is 0.03 mm during a motion with a maximum velocity of 0.2 m/s and 0.005 mm (1 pulse) when the motion is finished for the x-axis. For the y-axis, these parameters are 0.1 and 0,04 mm (1 pulse).

The results of the validation of motor model of (Kuppeveld and Sprik, 2006) concluded that the motor models available in the model libraries of 20-Sim are not the best models. The 20-Sim motor model is not port-based, i.e. it is only possible to control the motor with a current. It is recommended to have a port-based and configurable with details motor library and a reusable model of a DC motor controlled by a PWM signal.

The complete model of the ViewCorrect Plotter setup is validated with a maximum velocity of 0.2 m/s due to safety reasons. If the setup is safe enough as specified in section 2.1, the model should be revalidated with the specified maximum velocity of 0.5 m/s to obtain the new accuracy.

The next chapter describes the design of the plotter software.

4 Plotter Software

4.1 Introduction

The construction of ViewCorrect Plotter is ready to draw everything. However with the previous plotter software, generated by 20-Sim, the movements of the plotter were limited to motion profiles which are not suitable for drawings. It is needed to have a workflow which allows the user to make a drawing in a Computer Aided Design (CAD) drawing package and plot this drawing with the plotter.

4.1.1 Workflow output of the plotter

The workflow is depicted in Figure 4.1.

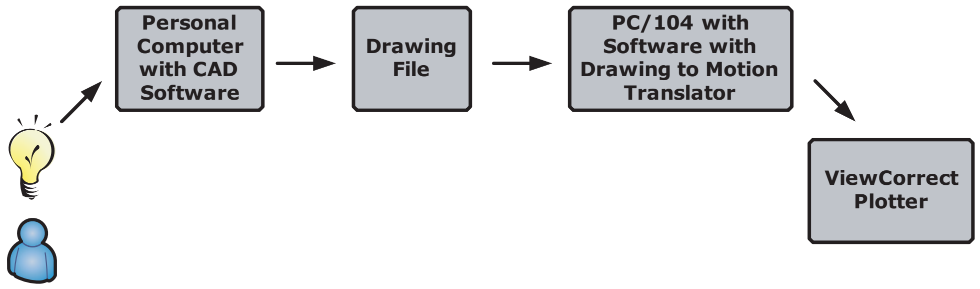

Personal Computer with CAD Software
Drawing File
PC/104 with Software with Drawing to Motion Translator
ViewCorrect Plotter

Figure 4.1: Workflow output plotter.

The PC on the left side is the starting point. The user has developed an idea for a drawing which has to be drawn by the plotter. The user designs his idea in a CAD drawing package. This idea will be stored in a file in a format which depends of the software tool used. This file contains all the information about the drawing. This file will be send to the PC/104. The PC/104 contains the plotter software. A part of this software is the control software. The control software controls the motors and other electronics. The input for the control software is the drawing to motion translator. This translates the file into the correct movements of the pen.

This chapter starts with the requirements and restrictions for the plotter software (4.2). This is followed with a analysis for a suitable drawing file (4.3) and a description of the structure of the drawing to motion translator (4.4). This chapter ends with the design of the plotter software in gCSP (4.5).

4.2 Requirements and restrictions plotter software

This section discusses the requirements and restrictions for the plotter software.

4.2.1 Requirements

The requirements for the plotter software are:

- Possibility to make the drawing in a CAD drawing package.
- Possibility to draw 'everything' the user wants. The word everything is between quotes, because it is needed to keep the restrictions of a plotter in mind. These restrictions are discussed in the next subsection.
- The software should be prepared to be used distributed over multiple computing nodes. In Figure A.6(b) a schematic overview of the setup with distributed control is depicted.

- The software should be designed according the top-down approach and tested with co-simulation.
- The drawing to motion translator discussed in section 4.4 and the controller discussed in section 3.4 should be embedded in the plotter software.
- The drawing to motion translator should limit the movements of the pen to the drawing area of the plotter, which is smaller than the end switch to end switch area.
- The software should contain a safety layer, which limits the steering signals to a safe domain. In case an end switch is hit, the safety layer should set the relevant steering signal to zero.

4.2.2 Restrictions

The ViewCorrect Plotter plots its output by moving a pen across the surface of a piece of paper. Consequently, the plotter is restricted to line art in contrast with raster graphics as with other printers. Solid filled objects can be drawn, but this requires exact knowledge of the pen point width. In case of a small pen point, this takes a lot of time. Consequently shaded filled objects are preferred. The plotter can draw complex line art, but at a low speed because of the inertia of mechanical construction of the x-, y- and z-axis. Drawing line art runs efficient with vector format drawings.

Vector format drawings are images that are described using mathematical definitions. This is in contrast to bitmap or raster format drawings. Raster images are described using pixel data. Every pixel contains the data for the corresponding colour. Most of the pen plotters use a vector based translator. Raster plotters can use a vector or a raster based translator. Consequently raster formats drawings are not useful for pen plotters.

4.3 Drawing file

4.3.1 File formats

As stated in the previous subsection, vector format files are suitable for plotters. Two kinds of vector format drawing files can be used. The first option is a file which contains all the information of the drawing. The second option is a file which represents the original drawing and contains all the information of the movements of the pen. Due to the existence of dozens of CAD drawing packages, a lot of different vector file formats exists. Appendix C shows a list with all common vector formats.

The first option means a file format in which the CAD drawing package stores the information of the drawing. The disadvantage of these formats is that they only contain information about the drawing, but not the content which describes how to make the drawing. Consequently choosing this option requires more work for the drawing to motion translator.

The second option means a plot command file format, which is generated from the original vector drawing format. Creating a plot file is a case of capturing the information that is normally sent to a plotter and saving it as a file instead. The file is produced by using standard plot/print tools within an OS. The advantage of plot files is that they represent the original drawing, including the order and movements of the pen. The most CAD drawing packages contain plot file generators. Appendix D.1 gives additional information about a plot file.

The best option here is to use the plot command file format. These files are as close a representation as possible to the paper. This is less work for the drawing to motion translator. Besides that they can be produced by the majority of CAD software tools and these files are more than ten times smaller than the normal CAD file formats. Plot files are written in a plot language. The next section describes the different plot languages.

4.3.2 Plot file languages

In the last decades multiple plot languages have been created. These languages contain commands with detailed information, for example draw a line from here to there. The de facto standard today is Hewlett Packard Graphic Language (HP-GL) or its successor HP-GL/2 (Oce, 2007). Other plotter manufactures created their own forms with extensions for their machines, for example Houston Instruments with Digital Microprocessor Plotter Language (DMPL). Another common language is Gerber used by Printed Circuit Board (PCB) manufactures. HP-Printed Command Language (HP-PCL) and HP-Raster Transfer Language (HP-RTL) are more advanced printer and/or plotter languages. In Appendix D the plotter languages are described in more detail.

Plot files in certain file formats can be viewed by special tools. The most common viewers read HP-GL(/2) files. Plot files in certain file formats can be converted to other file formats or edited. In this way plot files can be used to communicate about designs.

HP-RTL is a vector and raster plotting language and for the pen plotter only a vector plotting language is needed. HP-PCL is a more complicated language because of the supported features for laser printing. It uses HP-GL for the vector plotting. This leaves HP-GL, HP-GL2 and DMPL as options. All three languages describe the vector movements of the pen. HP-GL2 supports some extra features. DMPL is a form a HP-GL, but is less supported by CAD drawing packages, plot file viewers and the industry. This leaves HP-GL and HP-GL2 as options.

4.3.3 Conclusion

The implementation of the workflow as described in 4.1.1 is described here. The CAD drawing package generates a plot file written in HP-GL(2). This file can be verified for correctness with HPGLview (CERN, 2007), which is a HP-GL(2) viewer. This file is sent to the PC/104. A part of the plotter software is a drawing to motion translator. The drawing to motion translator is a HP-GL(2) translator program that supports all the commands that are needed. For instance support for different kinds of fonts may not be necessary, but commands to draw a line, move the pen up or down are obvious. This interpreter translates the HP-GL(2) commands to setpoints for the motors. The implementation of the drawing to motion translator is described in the next section. In Appendix E screenshots are depicted of the tools to illustrate the workflow.

4.4 Drawing to motion translator

The drawing to motion translator is a combination of a couple of functions, decisions and input, which generate an output. The input is a file. The outputs are setpoints. In Figure 4.2 the flowchart of the drawing to motion translator is depicted.

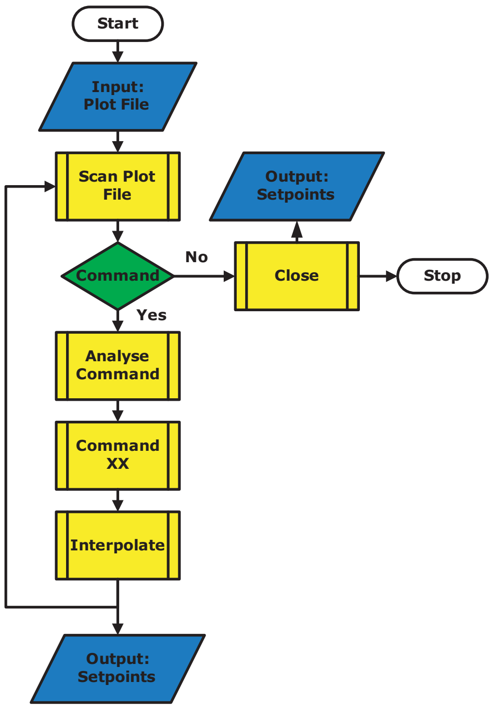


Figure 4.2: Flowchart drawing to motion translator.

Input: The input is a plot file written in HP-GL(2), generated by a drawing tool like AutoCAD or CorelDraw.
Scan Plot File: The plot file is opened and scanned on the occurrence of a semicolon, because every HP-GL(2) command is closed with a semicolon. On the occurrence of a semicolon, a command is found and the function *Analyse Command* is called. If no or no more semicolons are found the function *Close* is called.
Analyse Command: The founded text is cleaned, i.e. all non-HP-GL(2) characters like white space are removed. After the command is found, a specific command function is called.
Command XX: All the needed parameters from the HP-GL(2) command are given to the function *Interpolate*.
Interpolate: Setpoints are calculated in meters for the x- and y-motors. If the status of the z-as changes, the setpoints of the x- and y-motors are paused for a fixed time. In this way, the servo motor is able to move the pen up or down. The function *Interpolate* limits the setpoints to the maximal size of the drawing area for safety reasons. The setpoints for drawing a line are calculated according a cycloidal algorithm.
Close: Function *Close* calculates setpoints from the last setpoints back to the origin. After this function the drawing to motion translator is ready and stops.

Figure 4.3 shows the pattern of the motion profile of function *Interpolate*. The duration of such a motion profile depends on the length of the line and a predefined parameter of the maximum velocity. The advantage of cycloidal profile for the velocity is that at the end of the command the velocity is zero. Consequently, drawing right or acute angles have no risk of overshoot. However this is not always the optimal way of drawing looking at the time aspect. For example, when the next line is almost in the same direction the plotter still stops between both lines, instead of just moving on without slowing down the velocity. However finding the optimal way of moving is a project in itself.

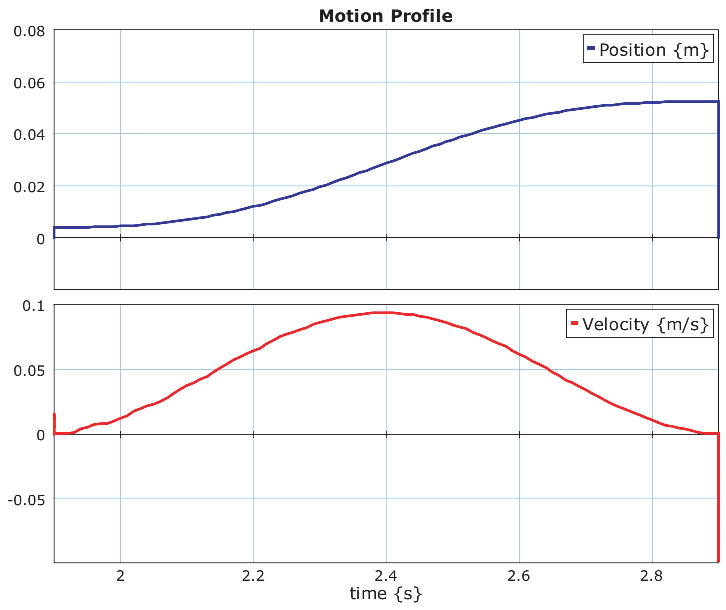


Figure 4.3: Motion profile of a line.

4.5 Plotter software in gCSP

The plotter software is written in gCSP. The plotter software runs at the PC/104. The software is written in gCSP, because of the ability to design real-time CT-based software in a graphical application.

This section describes the design of the plotter software in gCSP. First it gives an introduction to gCSP. This is followed by a functional top level design and ends with a top level structure in gCSP.

4.5.1 Introduction in gCSP

The gCSP tool gives a user the possibility to design a Communicating Sequential Processes (CSP) model graphically. CSP is a language in which concurrent systems can be described and analysed algebraically (Hoare, 1985). The structure of the design can be defined by compositional and communication relationships between processes, like a writer, reader or repetition etc. Subsequently by grouping these processes into constructs, like sequential, parallel or pri-parallel etc, the structure is complete. Communication with hardware is implemented with link drivers. From the gCSP model C++ code can be generated. This code uses the CT library

(Hilderink et al., 2000). The CT library is based on the concept of the Occam language, which is a parallel programming language and CSP principles.

Besides code for formal verification of the CSP model can be generated from the model. The formal verification of the model can be done in FDR2 (FDR2, 2007). FDR2 is a model checker which is capable of analysing the models against properties as for example deadlock and livelock.

4.5.2 Functional top level design

From the requirements in the previous section a functional top level design is made. This is depicted in Figure 4.4.

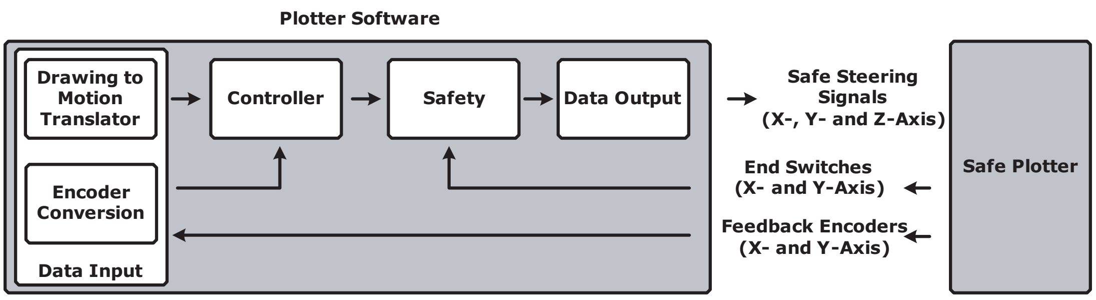
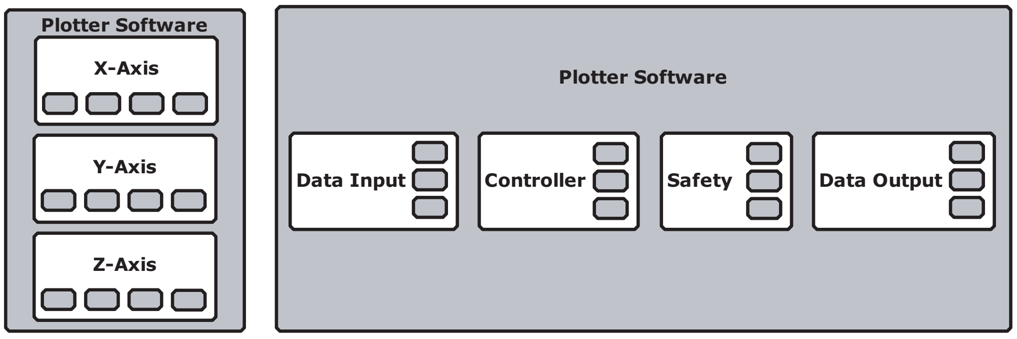

Plotter Software
Drawing to Motion Translator
Controller
Safety
Data Output
Encoder Conversion
Data Input
Safe Steering Signals (X-, Y- and Z-Axis)
End Switches (X- and Y-Axis)
Feedback Encoders (X- and Y-Axis)
Safe Plotter

Figure 4.4: Functional top level design plotter software.

DataInput reads the encoder values and converts the pulses to meters, and the drawing to motion translator calculates the setpoints for the motors. *Controller* compares the reference values with the converted feedback , calculates a steering signal and generates control signals for the z-axis. In *Safety* the steering values will be limited to a safe domain and the *Safety* component detects if an end switch is hit. The *DataOutput* component writes the data to the Anything I/O board, which sends the signals to the motors.

In order to be prepared for distributed control, a functional design does not seem to be a natural way of designing distributed software. It would be more logical to split the software into distributed components (X-, Y- and Z-axis) and divide these components into the functional components described in the previous paragraph. However a functional design gives a direct clear view of the structure of the software. In Figure 4.5 both views are depicted. The small grey blocks in the larger white blocks are in the left figure the functional components and in the right figure the distributed components. The dependencies between the components do not change in the different views.

It can be concluded that different views require different software architectures, but with the same specifications. Consequently, it would be useful to have the possibility to change between these views in gCSP. However with the current version of gCSP this is not possible. Therefore the plotter software is designed from the functional point of view, to have a clear view of the software and because distributed control will not be implemented in this project. In the next subsection the functional top level design will be translated to a CSP-based architecture.



(a) Distributed design view for the plotter software

(b) Functional design view for the plotter software

Figure 4.5: Two types of design views for the plotter software.

4.5.3 gCSP top level structure

In Figure 4.6 the top level structure of the plotter software in gCSP is depicted.

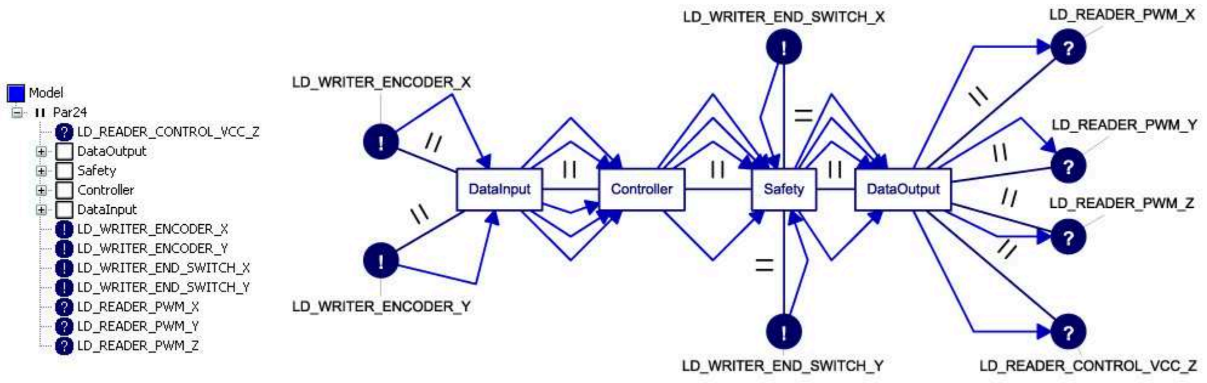


Figure 4.6: gCSP top level architecture of the plotter software.

The four components are implemented as processes. All processes work parallel in relation to each other. The details of the four processes are shown in Appendix F.

Timing is not yet included in the plotter software. Previous projects (Maljaars, 2006); (Deen, 2007) concluded that the timed execution of (control) processes in the CT-library contains too much jitter. The CT-library is designed with the basic idea that it is an OS in itself with an own scheduler and an own threading system. Consequently, all parallel processes are executed in one Linux thread. The current timing implementation contains a work around for the blocking system calls (Groothuis, 2004). Besides the rendez-vous principle of communication makes the timing behaviour unpreditable. It is recommended to analyse this first, before implementing time constraints in the plotter software. However with co-simulation, the plotter software can be synchronized with simulated timing from 20-Sim. This is discussed in Chapter 5.5. In this way, the software can still be tested functionally with simulated time. However not with effects like jitter in hardware timing or behaviour of the scheduler on timing.

4.6 Conclusion

A workflow is analysed and presented allows the user to make a drawing in a CAD drawing package and plot this drawing with the plotter. The plot command file format is chosen as vector file format which is send to the ViewCorrect Plotter setup. This format represents the original drawing which is designed in a CAD tool, including the drawing order and the movements of the pen. HP-GL(2) is chosen as plot language. A drawing to motion translator is designed which can read a plot command file, written in HP-GL(2), and translates the different commands to setpoints for the motors. The setpoints are limited to the drawing area of the plotter for safety reasons. The drawing to motion translator calculates the setpoints according a cycloidal pattern. At the end of a draw command the velocity is zero and in the middle maximum.

The plotter software is designed in gCSP with a top down approach and with a functional design view. It is divided in four processes: *DataInput* reads the encoders and the drawing to motion translator calculates new setpoints, *Controller* computes new steering values from the feedback and reference values, *Safety* limits the steering values to a safe domain and *DataOutput* writes these values to the Anything I/O board.

The plotter software is designed with a functional view. To be prepared for distributed control, it would be more natural to have a distributed design view. However, a distributed design view gives the software a less clear structure. In the current version of gCSP it is not possible to change between these views. It is recommended to have this possibility in gCSP.

No timing has been added to the plotter software. Previous projects concluded that the timed execution of (control) processes in the CT-library contains too much jitter. It is recommended to redesign timing dedicated for the Linux OS with real-time package instead of the current work around.

A cycloidal profile for the velocity is not always the optimal way of drawing looking at the time aspect. Future research is needed to find the optimal way of moving for the plotter.

The next chapter describes co-simulation. It ends with a case study of testing the plotter software together with the dynamic model of the ViewCorrect Plotter setup.

5 Co-Simulation

5.1 Introduction

The plotter software written in gCSP and the model that describes the behaviour of the View-Correct Plotter setup, designed in 20-Sim, are connected to each other. In this way, it is possible to test the software with its physical environment without using the real plant. In order to test the generated code from gCSP together with 20-Sim, a co-simulation facility has to be designed.

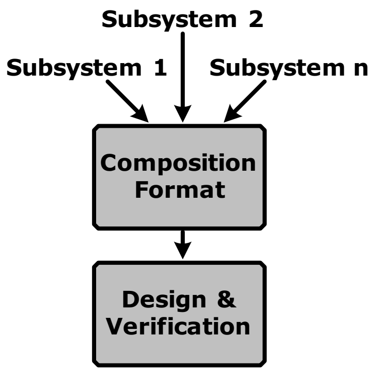
A lot of commercial co-simulation facilities are available, but they are dedicated to a specific design environment and only for connecting with another specific design environment. Currently no flexible framework exists for co-simulation. This project starts to address the requirements and analyse such a framework.

This chapter starts with discussing heterogeneous design approaches (5.2), details, requirements and challenges of co-simulation (5.3). This is followed with a co-simulation facility framework (5.4) and a case study of testing the plotter software together with the model that describes the behaviour of the ViewCorrect Plotter setup(5.5).

5.2 Heterogeneous design approaches

A mechatronic system, like the ViewCorrect Plotter, is an example of a heterogeneous system. It consists of components belonging to different domains. These components are being described using different specification languages. Like VHDL for hardware or C for software. Two approaches have been developed to design heterogeneous systems: the compositional approach and the co-simulation approach.

The compositional approach, Figure 5.1(a), tries to integrate the different parts into a unified representation. This representation is used to design and verify global behaviour. The unified representation makes its possible to do full coherence and consistency checking. New specification languages can be added to the composition format. A major disadvantage of this approach is that it does not give the design engineer the freedom to choose the most suitable design environment. Examples of this approach are Polis (Polis, 2007) and Ptolemy (Ptolemy, 2007).



(a) Compositional approach

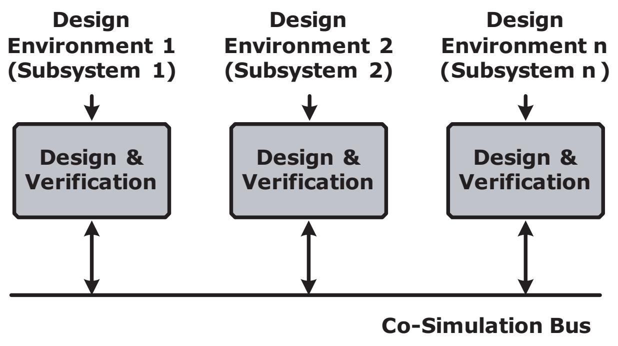

Design Environment 1 (Subsystem 1)
Design Environment 2 (Subsystem 2)
Design Environment n (Subsystem n)
Design & Verification
Design & Verification
Design & Verification
Co-Simulation Bus
(b) Co-simulation approach

Figure 5.1: Two types of approaches for heterogeneous design.

The co-simulation approach, Figure 5.1(b), connects multiple design environments to each other with a co-simulation bus. This allows for the use of the most suitable environments for each component. The success of this approach depends to a large extent whether the design environments allow for cooperation. Examples of this approach are VCI (VCI, 2007) and CoWare (CoWare, 2007). However both examples only deal with hardware/software co-design. Other examples like Data-Distribution Service (DDS) (DDS, 2007) and RT-LAB Orchestra (Cécile et al., 2006) only deal with distribution of the data, not with issues like continuous time or discrete event models. A lot of commercial packages are available to connect only two design environments, like PSpice-Matlab/Simulink (EMA, 2007). A difficulty of the co-simulation approach is the checking for overall coherence and consistency due to the fact that the system is split up in multiple subsystems. In the next section, co-simulation is discussed more extensively.

5.3 Co-simulation

Co-simulation is simulation of a heterogeneous system of which parts are interacting and distributed over more than one simulation engine connected with a co-simulation bus. In fact it is a joined simulation of the different parts of the system by using simulation engines and abstraction levels appropriate to each part. It can be used between different design environments, describing different domains or components and between different PCs.

A major reason to use co-simulation is that it allows for combining separate design groups, like software and control engineering. This results in a heterogeneous system design and verification environment, instead of only the design and verification of the separate part or domain. This design and verification can be done at different abstraction levels, so it is a verification technique in the entire design cycle. Other additional advantages are (re-) using of models developed in other design tools. Increase of simulation speed when simulation takes places at multiple processors. Besides it allows for cooperation of design engineers at different places. Two types of co-simulation are developed: untimed and timed simulation.

5.3.1 Untimed and timed simulation

Untimed co-simulation considers the functional behaviour of the overall system. Untimed co-simulation is event based. The data exchange between the design environments is controlled by events. Untimed co-simulation only verifies sequences of operations. Timed co-simulation considers both functionality and (real-) timed behaviour, now timing aspects are included in the functional behaviour. The major problem for timed co-simulation is synchronization between the different design environments.

Untimed or timed co-simulation can be used at different stages of a design cycle. At the beginning untimed co-simulation can be used to verify the functional behaviour of the high level definition of the various components or functions of the system. Later in a design cycle, (real-) timed co-simulation can be used to verify the behaviour of the refined description of the components.

5.3.2 Requirements

The co-simulation facility is described with the following requirements. A major requirement is the first:

- The results of co-simulation should be same as when the different parts where modelled in the same design environment. Consequently the co-simulation facility should not have any effect on the results of the simulation.
- An interface definition, how to connect to the design environment to the co-simulation bus, and communication protocol, how to send and receive data across the co-simulation bus, is needed.

- The design environment interface should be generated automatically. This allows the user, regardless of communication protocol or architecture, to concentrate on simulation. Besides it will be easy to use for engineers of other disciplines, because it requires no knowledge of other design environments.
- The data communication should be synchronized, especially during timed co-simulation. All design environments should be at the same point of execution of the overall system model.
- A Graphical User Interface (GUI) or 3D animation model is needed to visualize the impact of design decisions to other design disciplines and to observe problems.
- Able to perform the co-simulation distributed at different workstations/places.
- A common parameter database is needed to strengthen the value of the co-simulation process. In this database all (common) variables/constants are stored for checking purposes.

The requirements mentioned in the previous section lead to a couple of challenges.

5.3.3 Challenges

Model computation

Computation of the parts might become a problem, when they are simulated with different step sizes or consist of parts simulated in the continuous time or discrete event domain. If all parts use the same step size for their computation, no problems exists related to data availability. However this is not always possible. If some part calculates its state at a lower step size as the other interacting parts, the other parts need data interpolation in this step to avoid erroneous simulation. Figure 5.2 shows an example of two parts, where one simulation process operates at a higher step size and needs data interpolation.

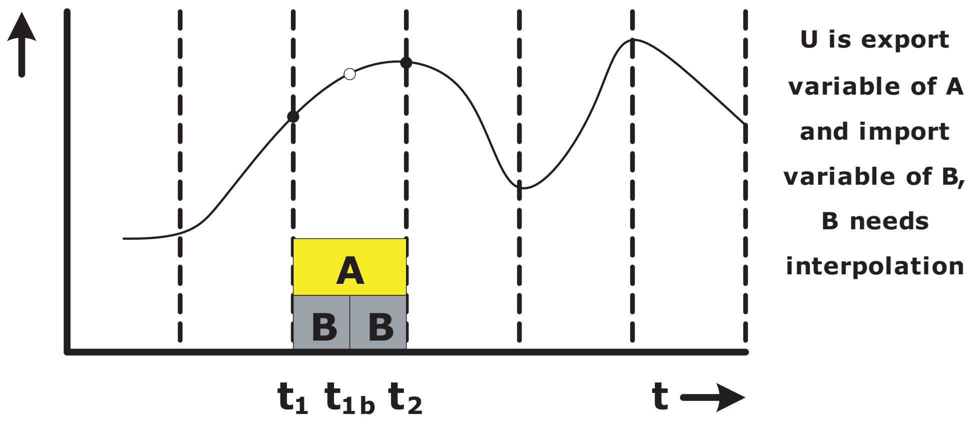

A
B
B
t_1
t_{1b}
t_2
t
U is export variable of A and import variable of B, B needs interpolation

Figure 5.2: Two simulation processes with different step size.

Data interpolation

Variable step sizes introduce the need for data interpolation. In order to interpolate variables, the rate of change should be known. This requires a storage buffer to obtain this information. Different data interpolation methods, like Shannon (Wikipedia, 2007b) or Kriging (Wikipedia, 2007a) exists.

Continuous time and discrete event

A design environment can work with a discrete event model or a continuous time model. In continuous time models, time is a global variable and advances by (variable) integration steps. In discrete event models, time is a global notion for the overall system. It advances discretely at the occurrence of an event. Continuous time models are computed as discrete time models. Synchronization of these different time models might be problem. Besides, the discrete event simulator must detect state events. "A state event is an unpredictable event, generated by the continuous time simulator, whose time stamp depends on the values of the state variables (for example: a zero-crossing event or a threshold overtaking event)" (Gheorghe et al., 2007). At detecting a state event, the discrete simulator has to advance until the time of the state event, instead of advancing with a normal simulation step. In Nicolescu et al. (2007) the operational semantics for synchronization in continuous/discrete models are presented. The basic idea is illustrated in Figure 5.3. It uses the Discrete EVent System Specification (DEVS) formalism (Zeigler and Kim, 2000). This is an abstract simulation mechanism and enables event-based, distributed simulation. The implementation for a co-simulation interface of this formalism is presented in (Gheorghe et al., 2007).

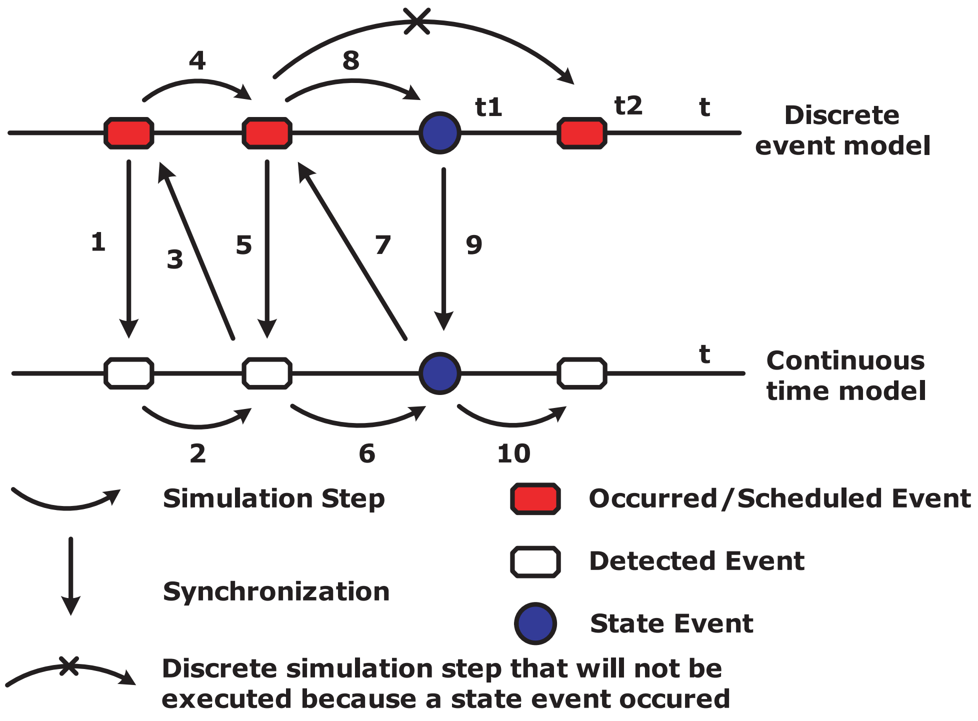


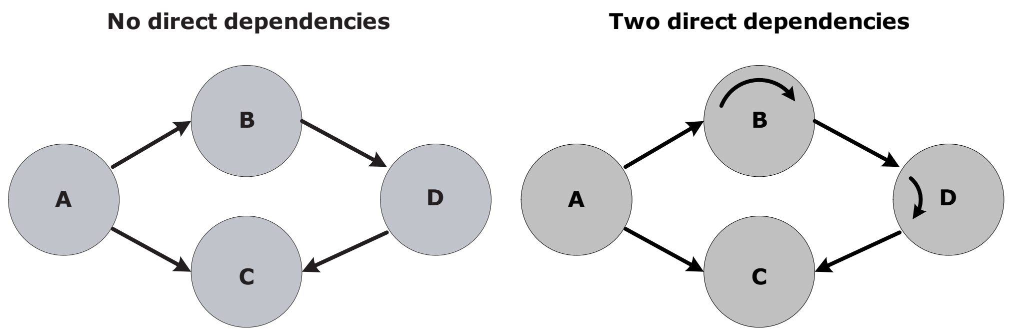
Figure 5.3: Continuous and discrete-event models synchronization (Nicolescu et al., 2007).

The discrete event simulator starts with executing all processes and updates all signals sensitive to the notified events at zero time. The continuous time simulator gets a time stamp of the next output event of the discrete event simulator (1). The continuous time simulator advances to the time stamp (2) and switches to the discrete event simulator (3), which advances to the event time stamp (4) and this cycle restarts again.

The continuous time model might generate a state event. Now the continuous time simulator detects this state event and gives this time stamp to the discrete event model and switches to the discrete event simulator (7). The discrete event model advances to the time stamp and executes all processes that are sensitive to this external event (8). An advantage of this model is that it does not need a rollback if a state event is detected.

Model ordering

The order of computation of the different parts might become a relevant issue. A part might have import and export variables. If the export variables only depend of their own states and time, the order of computation is not important. This is depicted in Figure 5.4(a). The different parts can be calculated in parallel in random order. However if a direct relation between import and export variables exists, it is needed to have specific order of computation. This is depicted in Figure 5.4(b). Now the different parts can only calculated in a certain order.



(a) No direct dependencies (b) Direct dependencies

Figure 5.4: Dependencies between simulation processes.

The relations between the different parts might result in an algebraic loop. This can only be solved by iterative methods. If the algebraic loop can not be eliminated, the only solution is to use an algebraic loop solver. This forces the simulation engine to do a lot more computations.

It is needed to know the different dependencies between the processes, a dependency graph. This dependency graph is used to investigate the overall system with respect to the order of computation. An automatic solution for a dependency graph, only looking at the import and export variables, is not possible. The rate of change of a export variable investigated with respect to the rate of change of the import variables might give the idea of a direct relation. However the cause may be a non-linearity of the component. A integrated solution for a dependency graph would be a translation of the part into a formal specification language. The combination of these code blocks, from the different design environments, represent the dependency graph. A similar approach is used in gCSP models and FDR (see subsection 4.5.1).

5.4 Co-simulation facility framework

From the requirements, a framework is chosen to handle the described challenges. The co-simulation facility uses a client/server approach. The co-simulation server is a separate unit which controls the overall simulation process. The design environments are clients and exchange data via the co-simulation interface and the co-simulation bus with the co-simulation server.

The reason to choose for a client/server approach is the fact that the model ordering and the parameters relation file requires the need for an extra program, which controls the overall simulation process. Putting the model ordering and parameters relations processes in the interfaces requires more manual activities, to know for example the interacting with other design environments, because the overall system information is missing. With an extra program in the co-simulation server the design environments and the interfaces need no extra intelligence and therefore the interfaces can be generated easier automatically. A disadvantages of putting all the intelligence in the server is that this creates a higher data transfer rate, because now all the data, for example due to data reconstruction, needs to be send to the clients.

Due to the fact that the co-simulation facility should be able to be distributed over different workplaces, the co-simulation bus will be implemented over a Local Area Network (LAN) or even a Wide Area Network (WAN) and uses Ethernet as data link layer and Internet Protocol (IP) as network layer protocol. Two common transport protocols can be used; Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). In Appendix J both protocols are discussed extensively.

The main advantage of TCP is its own correction protocol; data will be send and faults are corrected. The disadvantage is that no knowledge exists about how much time it takes to repair faulty transmission. Consequently, it is not known how long it takes to send a packet of data. Another disadvantage of TCP is that broadcasting messages is not possible. UDP has no correction protocol, data will be send, but it is not checked if the data is arrived. An advantage is that faster transmission speeds and broadcasting messages are possible.

The best option here is to use TCP for the untimed and timed simulation, because of the reliability and the simulation time requires no real-time demands. In case of a real-time simulation TCP can be used, but that depends on the time constraints. Otherwise UDP has to used, because an own implementation of flow control is possible, which allows for higher real-time constraints. However along with UDP, it is needed to have error correction to handle for example missing or duplicate data packets.

The co-simulation facility framework is depicted in Figure 5.5. The components of the co-simulation facility framework are described in section 5.4.1.

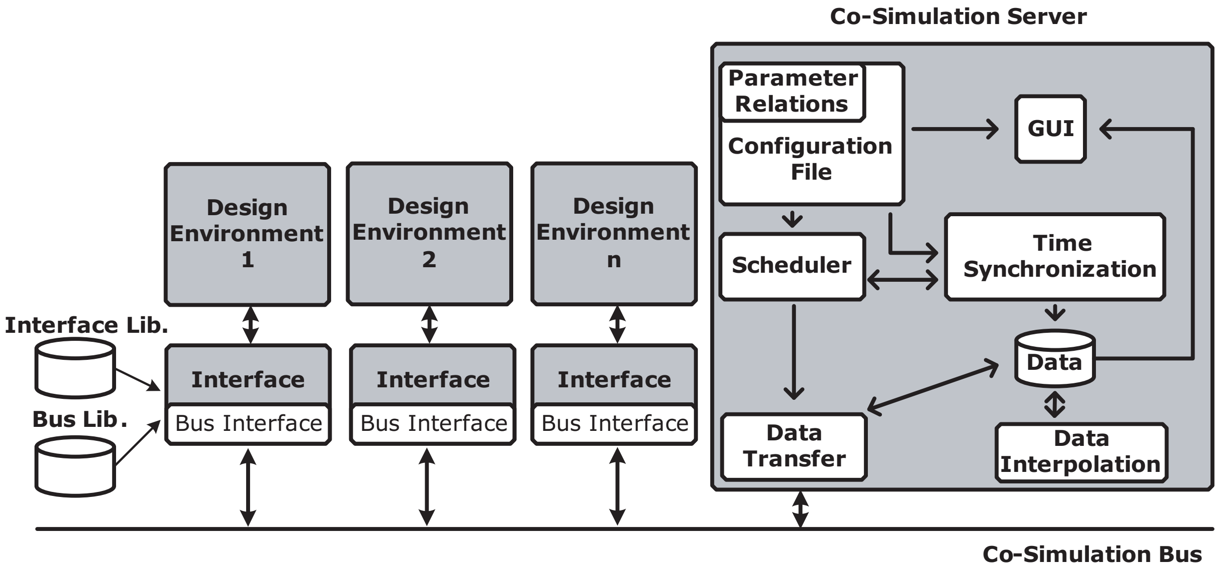


Figure 5.5: Co-simulation facility framework.

5.4.1 Description of components of the co-simulation facility framework

Bus Interface: The bus interface is part of the interface, but only takes care of exchanging data with the bus. The implementation depends on the technical implementation of the bus.
Bus Library: The bus library contains all the functions or processes to connect and exchange data between the interface and the co-simulation bus. The functions support the communication protocol and are specified to technical aspects of the bus.
Configuration File: This file is developed form high level system analysis and contains the architecture of the heterogeneous system. From the architecture the dependencies of the different subsystems are derived. This information is transferred to the sequence controller. The configuration also contains all common information about the simulation of the sub models, like simulation step size, continuous or discrete time. This is needed to initialise the timing component. All these information can be displayed at the GUI.
Co-Simulation Bus: The co-simulation bus transfers all data to and from the interfaces and server.
Co-Simulation Server: The server is in charge of the overall process. The server decides the order and synchronization of computation and interpolates data if necessary. Besides it checks if common variables of the subsystems are similar.
Data Storage: In order to interpolate data, the values of earlier moments have to be stored. These values will be stored at the server.
Data Transfer: Data Transfer transfers all data to the design environments and from the data storage according the communication protocol.
Design Environments: The design environment is software application suitable to do the design and verification of the subsystem.
Graphical User Interface (GUI): The GUI is the visualization of the overall system. The GUI gets the data to visualise the overall system from the data storage. If a design environment is capable of providing a visualisation of the overall system, it might not be needed to have a animation of the system on the GUI. Information from the configuration file and from the parameter relationship component can be displayed at the GUI.
Interface: The interface provides a connection between the design environment and the co-simulation bus.
Interface Library: The interface library contains all the functions or processes to connect and exchange data between the design environment and the bus-interface. The interface library is not aware of technical aspects of the bus. The implementation of the interface will depend on the structure of the design environment.
Parameters Relation File: This file contains all the variables/constants of the system and is part of the configuration file. In case the subsystems share variables, before the start of the simulation it will be checked if they are similar. Mismatches or other information can be displayed at the GUI.
Scheduler: From the dependency graph, the order of how the system has to be calculated becomes clear. In this order the different design environments will be executed. The scheduler commands the data transfer.
Time Synchronization: This component controls the time synchronization between the subsystems. It calculates the central time and the next time stamp. The sequence controller and the data storage component use this time stamp as a new endpoint for the simulation. If the data storage does not have the data at the specific time stamp, for example due to a different step size, then data interpolate will be commanded to calculate the missing points.

5.5 Case study: Co-simulation with 20-Sim and gCSP

This case study will give more insight in the value of the co-simulation facility framework. It should be clear that this is a long term project and that this case study is a start of the complete co-simulation facility. The main purpose is to test the plotter software, but it also will look at the co-simulation facility in a heterogeneous design view.

The idea is that the plotter software, discussed in Chapter 4, and the I/O and plant part of the model, discussed in Chapter 3, do not need to be changed for co-simulation. However with the current version of 20-Sim it is needed to use a Dynamic Link Library (DLL) to export or import data. The DLL has to contain predefined functions which are supported by 20-Sim. The plotter software reads and writes data from and to the hardware with link drivers. Therefore in gCSP it should be possible to replace the implementation of the link drivers with TCP/IP drivers instead of the Anything I/O board link drivers. In (ten Berge, 2005) such kind of link drivers have been designed. However tests concluded that these network and remote link drivers do not work under Windows. Therefore it is decided to start from scratch and write new cross platform link drivers for UDP or TCP communication over IP.

The bus interface is implemented with a *Socket* and a *Linkdriver* class. The network and remote link drivers' structure is not yet implemented in these classes. In *Socket* the basic functions for communication are implemented. The *Linkdriver* acts as a server or a client and uses functions from *Socket*. The co-simulation server is not implemented in this case study. Because of the fact that timing is not included in the plotter software, it is synchronized with timing from 20-Sim. Therefore 20-Sim acts as a server and gCSP as client. The co-simulation bus uses TCP as transport protocol. Timing is not included in the plotter software. If this is included, timing can be simulated with SimTimers (ten Berge, 2005). However it should be investigated if the simulated timing behaviour is conform the real timing in the CT-library.

In Figure 5.6 the framework of the case study is depicted. The plotter software sends steering signals applied as PWM values to 20-Sim. These values are converted to the analogue domain and sent to the plant model. In the plant model the motors, controlled with the PWM signals, are moving the plotter construction. The values of the encoders and end switches are given to the plotter software. The co-simulation server is not implemented in this case study and therefore circled with a dotted line. Now 20-Sim acts as a server.

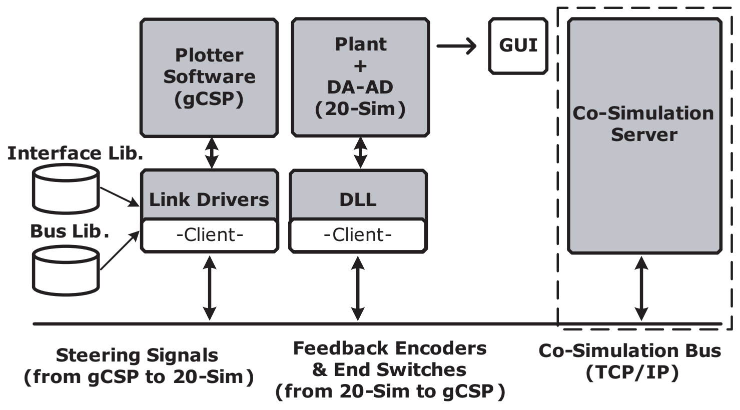
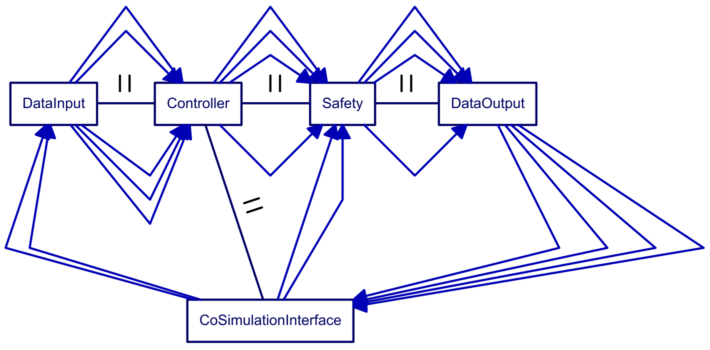
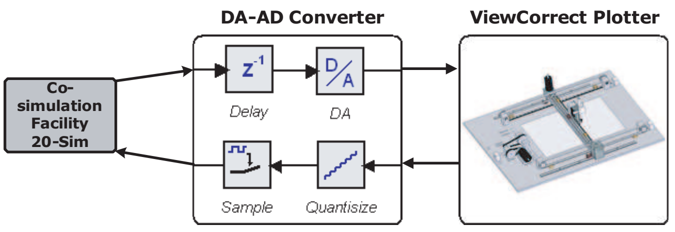


Figure 5.6: Schematic view case study: Co-simulation with 20-Sim and gCSP.

In the following figures, the gCSP and 20-Sim models are depicted with their co-simulation facility. The network and remote link drivers' structure is not yet implemented. Therefore an extra process is needed in the plotter software. This process called *CoSimulationInterface* replaces the link drivers as depicted in Figure 4.6.



(a) gCSP model

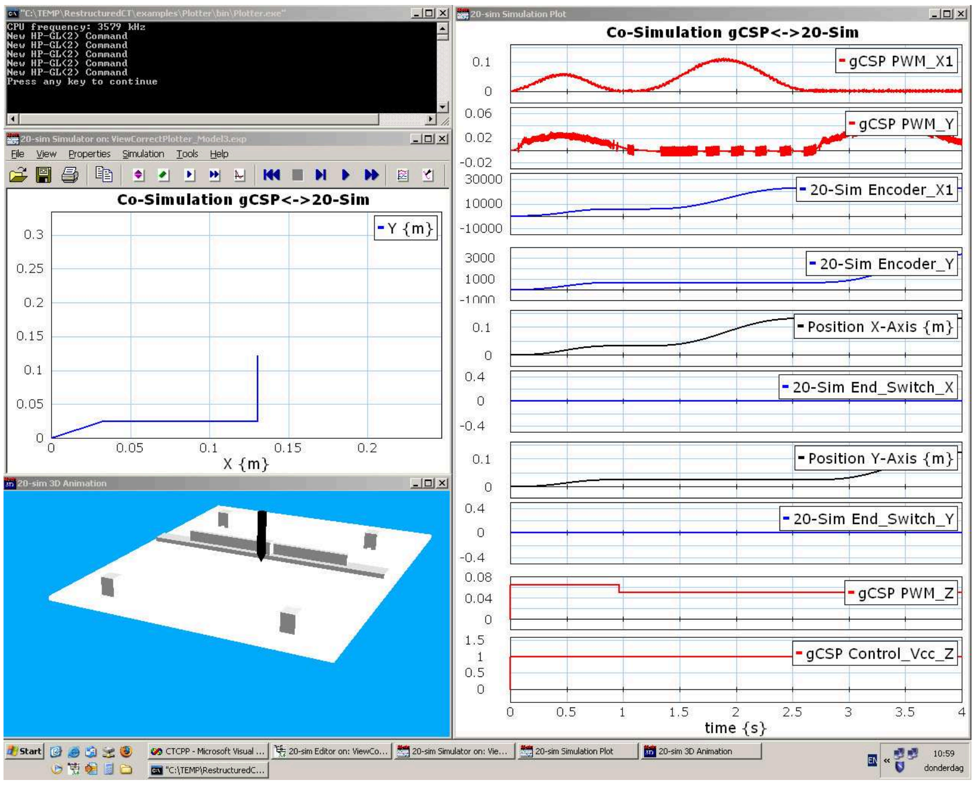


(b) 20-Sim model

Figure 5.7: Co-simulation facilities in both design environments.

5.5.1 Results

Two tests are performed for the plotter software. A functional test with simulated time under normal operation circumstances, which shows the correctness of the drawing and control part. The second test was a functional test with simulated time with deliberate disorder to show the correctness of the safety part. Figure 5.8 shows a screenshot of all the programs running. In the left upper corner the plotter software is shown. On the right side a 20-Sim simulation plot of the transfered and additional signals. On the left lower side the 3D animation model is depicted. Above the animation, an x-y simulation plot shows the movements of the plotter. Figure 5.9 shows the correctness of the safety part in case an end switch is hit. The plotter software reads the status of the end switch and with one timestep delay after the end switch is hit, the PWM signal becomes zero.



```
CPU frequency: 3579 kHz
New HP-GL<2> Command
New HP-GL<2> Command
New HP-GL<2> Command
New HP-GL<2> Command
New HP-GL<2> Command
Press any key to continue
```

Figure 5.8: Screenshot co-simulation with 20-Sim and gCSP.

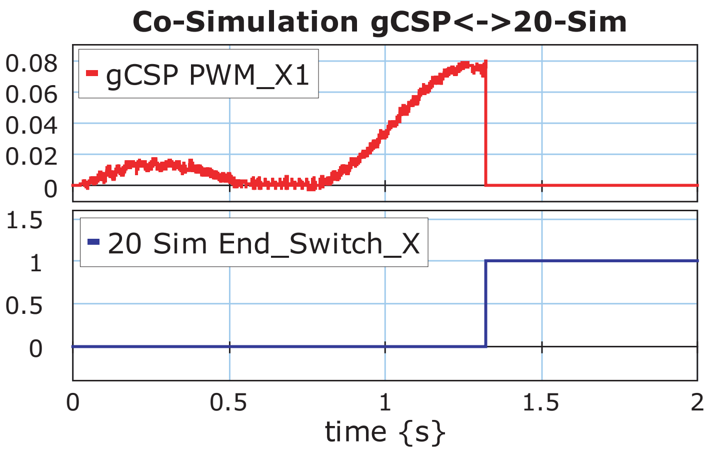


Figure 5.9: Case study: Testing safety part plotter software.

In these tests, a couple of minor and critical code errors where discovered during co-simulation. Now the errors could be fixed without using the real setup. Figure 5.10 shows one of these errors due to a wrong implementation of an if statement. It happened in the safety part and caused the PWM signal to reach his maximum (± 0.9) value in certain conditions instead of staying zero because an end switch is hit. On the real setup this could have caused hazardous situations.

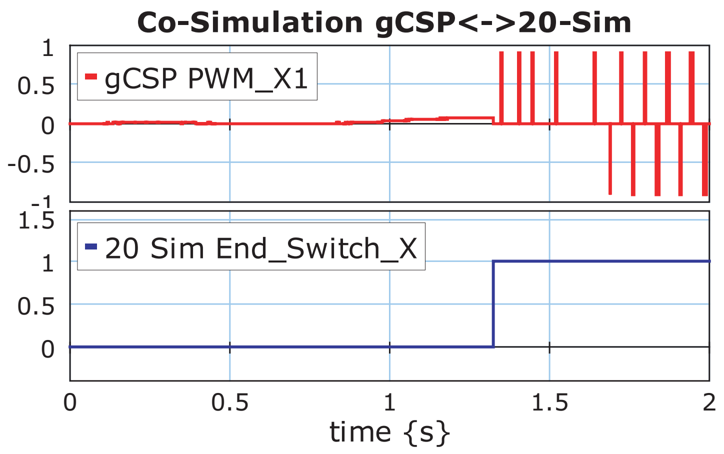


Figure 5.10: Case study: Critical code error in plotter software.

5.5.2 Conclusion of the case study

The strength of co-simulation for testing software has been proved by the fact that the functional correctness can be verified and errors can be fixed before running the software on the real setup. In this way the 'first time right' principle of developing software becomes an easier goal, because now the software can also be verified on functional and timed behaviour. However the results depends on the quality of the model. An incorrect or not accurate enough model can give the idea that the software might be right while this is not the case. Co-simulation allows for concurrent engineering, because the software can be developed while the setup might be still under construction. Another advantage is that co-simulation does support rapid prototyping, because features can be verified in different design environments without the need to build the setup. It can be concluded that co-simulation is a powerful tool in heterogeneous system design and especially for testing embedded software in a model-driven design approach, which brings engineers from different disciplines in a natural way together.

This case study has shown the necessity for a configuration file, because both simulators have to operate at a same simulation frequency. Now this has to be edited manually which is susceptible for human faults. Automatic interface generation is not included in the current facility, although it is rather straightforward to implement. Looking at this case study, it is recommended to have automatic interface generation for different design environments, because each design environment has its own structure for cooperation with extern tools. Another fact that was illustrated during the case study is that different names belonging to the same variables or constants require good communication agreements. However a configuration file and a parameter relation file, as stated in the co-simulation facility framework, can overcome these challenges.

5.6 Conclusion

Co-simulation has been analysed in the scope of heterogeneous system design. The requirements for and the challenges related to co-simulation have been investigated. A flexible co-simulation facility framework is designed. A case study is performed to evaluate co-simulation. In this case study, the plotter software, designed in gCSP, is tested together with the 20-Sim model of the ViewCorrect Plotter setup. In this way, the plotter software is verified on functional and timed behaviour without using the real setup. It can be concluded that co-simulation is a powerful tool for verification in a model-driven design approach for embedded control systems, which brings engineers from different disciplines in a natural way together. However the success of co-simulation depends on the quality of models used for testing, like the model used for testing the software, and whether the design environments allow for cooperation.

The next chapter describes a systematic workflow to isolate and solve causes of unexpected behaviour at small mechatronic setups, like the ones occurred during this project.

6 Failure Analysis

6.1 Introduction

A couple of times the ViewCorrect Plotter setup demonstrated unexpected behaviour. Because of the fact that the setup consist of several components like electrical, hardware and software, analysing the cause of this behaviour is not easy. Besides, the behaviour occurred at a time a new software application of the ForSee toolchain was used. This challenge requires a systematic workflow of verifying (individual) system performance characteristics. Such a workflow for the used components is not yet available at the CE laboratory. In the next section a systematic workflow is presented to isolate and solve a cause of unexpected behaviour. The workflow uses parts of the Failure Mode, Effects and Criticality Analysis (FMECA) approach as presented in Blanchard and Fabrycky (Blanchard and Fabrycky, 2004). This approach is translated to an workflow which is useful for failure analyses specialized at research setups at the CE laboratory. The words fault and failures are used as defined in (Jovanovic, 2006). A fault in a system is a defective value in the state of a component or in the design of a system. Failure is the behaviour of a system that deviates from that which is specified.

6.2 Systematic workflow

Finding a cause of a failure is finding how the failure is introduced in the system. Further it is needed to note the effect on other elements of the system and the system as an entity. This asks for identifying possible faults in the system, determine the causes of failures, determine the consequences of failures and identify failure detection means. In order to design and construct reliable systems, this failure analysis has to be done from the beginning of a project (”before the fact”). However, due to lack of time or bad system engineering this is not always implemented and happens ”after the fact”.

In general a research setup at the CE laboratory consists of components of different domains; mechanical and electrical engineering, and a ECS. The mechanical part is often a construction, where a subpart is able move in order to do a specific task. The electrical part is divided into two subparts. Firstly the actuators and sensors, secondly the PCBs to process the signals to the actuators and from the sensors to a I/O interface of the ECS. The ESC processes the available information from the sensors to steer the actuators. The ECS consist of hardware and software. The software is designed and created on a development PC with software applications, like 20-Sim and/or gCSP. The software is adjusted for and send to the hardware with other software applications, like the ForSee toolchain. In each of these domains failures might occur. Figure 6.1 shows the systematic workflow of failure analysis.

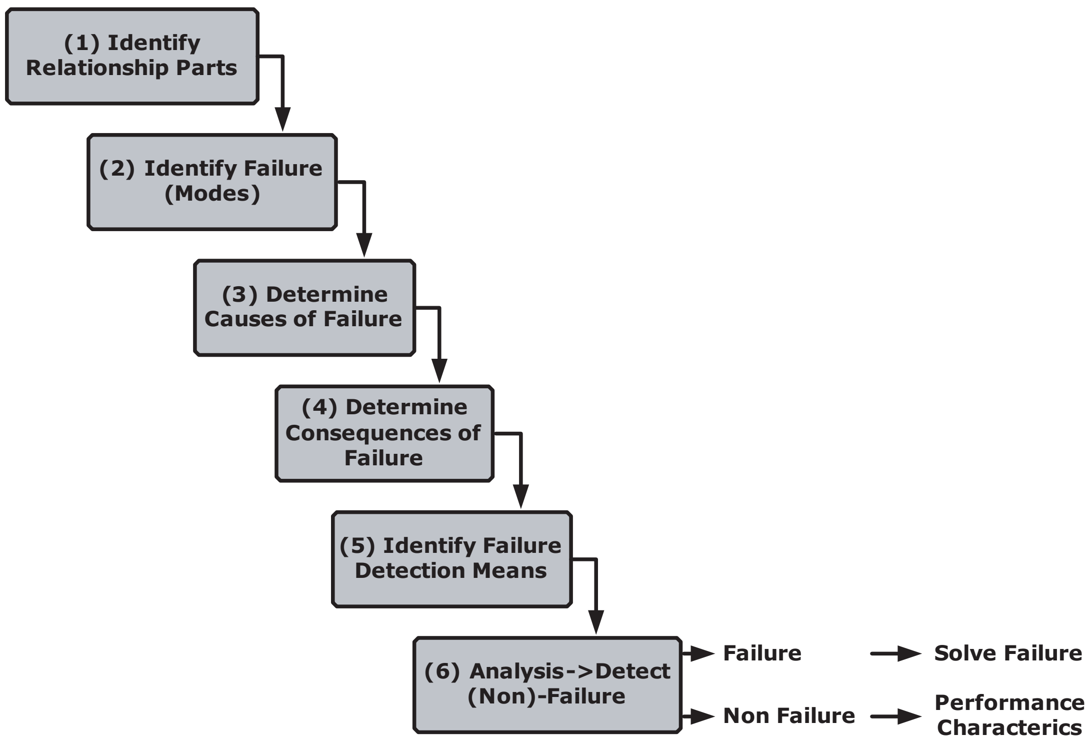

(1) Identify Relationship Parts
(2) Identify Failure (Modes)
(3) Determine Causes of Failure
(4) Determine Consequences of Failure
(5) Identify Failure Detection Means
(6) Analysis->Detect (Non)-Failure
Failure
Solve Failure
Non Failure
Performance Characterics

Figure 6.1: Systematic workflow for failure analysis.

1. Identify the relationship between the different parts and/or levels of the system. This helps with getting a global view and finding the effects of failure.
2. Identify, which failures are likely to happen. This can be extended with the failure mode, which is the manner a system element fails to accomplish his function. For example a sensor may fail cause of low power supply or dirt.
3. Determine the cause of the failure. This means analysing the environment, equipment, material or procedures. Examples of causes are a software coding error, defective materials or abnormal equipment stresses during operation.
4. Consider the consequences of the failures on the same, higher and overall system level. The results of step one can be helpful here.
5. Address ways to detect identified failures in the setup, with for example aids, tools or measurements devices.
6. Analyse the identified failures with the detection options described in step five. This step results in detection of a failure or detection of non-failure. In case of a failure, plans have to be made to solve the failure. In case of a non-failure, performance characteristics have to be reported. This starts with analysing the lowest level components of the system. After that the higher level components are analysed together with the lower level components or alone. The advantage of this approach is that in this way components from lower levels can be used for detecting failures at a higher level. This gives the advantage of using the existing components instead of making new components only for test purposes.

In ideal system engineering, these analyses are a part of the design. In small and not so complex projects, like the ViewCorrect Plotter setup, failure analysis is a supposititious part. Failures analyses occur not until in case of unexpected behaviour or failure. In this case the results of step four are outweighed by the amount of time it takes to fully analyse this. Consequently, it is tactful to forward to step five. In Appendix I, a case study of the ViewCorrect Plotter is demonstrated.

6.3 Conclusions

A systematic workflow is presented to isolate and solve causes of unexpected behaviour, like the ones occurred during this project. It uses parts of an existing approach, but is translated to a workflow for failure analyses at small mechatronic setups.

The failure analyses workflow is created during this project after unexpected behaviour occured. However, it has not been evaluated in other projects. It is valuable to evaluate the workflow for a new research setup during system engineering.

The next chapter discusses the conclusions and recommendations of this MSc-project.

7 Conclusions and Recommendations

7.1 Conclusions

7.1.1 Modelling and validation

The existing model of the ViewCorrect Plotter setup is adjusted and validated. The difference between simulated and measured parameters is less than a maximum of five percent. The 3D animation model is made for enabling a quick insight to verify the dynamic model as a whole and for demonstrating purposes. Controllers for both axes have been designed according the Ziegler-Nichols method. The accuracy of the model is 0.03 mm during a motion with a maximum velocity of 0.2 m/s and 0.005 mm (1 pulse) when the motion is finished for the x-axis. For the y-axis, these parameters are 0.1 and 0,04 mm (1 pulse).

7.1.2 Plotter software

A workflow is analysed and presented allows the user to make a drawing in a CAD drawing package and plot this drawing with the plotter. The plot command file format is chosen as vector file format which is send to the ViewCorrect Plotter setup. This format represents the original drawing which is designed in a CAD tool, including the drawing order and the movements of the pen. HP-GL(2) is chosen as plot language. A drawing to motion translator is designed which can read a plot command file, written in HP-GL(2), and translates the different commands to setpoints for the motors. The setpoints are limited to the drawing area of the plotter for safety reasons. The drawing to motion translator calculates the setpoints according a cycloidal pattern. At the end of a draw command the velocity is zero and in the middle maximum.

The plotter software is designed in gCSP with a top down approach and with a functional design view. It is divided in four processes: *DataInput* reads the encoders and the drawing to motion translator calculates new setpoints, *Controller* computes new steering values from the feedback and reference values, *Safety* limits the steering values to a safe domain and *DataOutput* writes these values to the Anything I/O board.

7.1.3 Co-simulation

Co-simulation has been analysed in the scope of heterogeneous system design. The requirements for and the challenges related to co-simulation have been investigated. A flexible co-simulation facility framework is designed. A case study is performed to evaluate co-simulation. In this case study, the plotter software, designed in gCSP, is tested together with the 20-Sim model of the ViewCorrect Plotter setup. In this way, the plotter software is verified on functional and timed behaviour without using the real setup. It can be concluded that co-simulation is a powerful tool for verification in a model-driven design approach for embedded control systems, which brings engineers from different disciplines in a natural way together. However the success of co-simulation depends on the quality of models used for testing, like the model used for testing the software, and whether the design environments allow for cooperation.

7.1.4 Failure analyses

A systematic workflow is presented to isolate and solve causes of unexpected behaviour. The workflow should be a part of the system engineering, but can also be used in case unexpected behaviour occurs. The workflow uses parts of the FMECA approach as presented in (Blanchard and Fabrycky, 2004), but is translated to a workflow which is useful for failure analyses specialized at small mechatronic setups like the research setups at the CE laboratory.

7.1.5 ViewCorrect Plotter setup

Pen mechanism

The pen mechanism has been redesigned and implemented. This mechanism can replace the pen easily and is prepared for future projects with another device like a milling cutter or inkjet head.

FPGA configuration file

The existing FPGA configuration file, available at the CE laboratory for the Anything I/O board, is extended with possibility to generate multiple PWM signals with different frequencies to be able to control the pen height.

Linear encoders

On both axes linear encoders are implemented. For this purpose different options have been analysed on costs, construction time, accuracy and robustness.

Printed Circuit Board for plotter I/O

A PCB is designed and realized to process the plotter I/O and named as Motor Control Block (MCB). The MCB has the same form factor and is stackable with the existing CE H-bridge PCB (van den Berg, 2006) to a complete electronic circuit to control a DC motor with an I/O board. The board is equipped with an additional I/O connector for measuring or extension with additional electronics. The additional connector is interconnected with the connector to the I/O board. Consequently, it is possible to measure the signals at the I/O board during operation.

7.2 Recommendations

7.2.1 Modelling and validation

The results of the validation of motor model of (Kuppeveld and Sprik, 2006) concluded that the motor models available in the model libraries of 20-Sim are not the best models. The 20-Sim motor model is not port-based, i.e. it is only possible to control the motor with a current. It is recommended to have a port-based and configurable with details motor library and a reusable model of a DC motor controlled by a PWM signal.

The complete model of the ViewCorrect Plotter setup is validated with a maximum velocity of 0.2 m/s due to safety reasons. If the setup is safe enough as specified in section 2.1, the model should be revalidated with the specified maximum velocity of 0.5 m/s to obtain the new accuracy.

7.2.2 Plotter software

The plotter software is designed with a functional view. To be prepared for distributed control, it would be more natural to have a distributed design view. However, a distributed design view gives the software a less clear structure. In the current version of gCSP it is not possible to change between these views. It is recommended to have this possibility in gCSP.

No timing has been added to the plotter software. Previous projects concluded that the timed execution of (control) processes in the CT-library contains too much jitter. It is recommended to redesign timing dedicated for the Linux OS with real-time package instead of the current work around.

A cycloidal profile for the velocity is not always the optimal way of drawing looking at the time aspect. Future research is needed to find the optimal way of moving for the plotter.

7.2.3 Co-simulation

A part of the designed co-simulation facility has been evaluated in the case study. The case study emphasizes the need for a configuration file and automatic interface generation. Design environments have different structures for cooperation with external tools. Therefore it is needed to have automatic interface generation specialized for a design environment. It is recommended to implement these issues in future research, because now this is done manually and therefore sensitive for faults.

7.2.4 Failure analyses

The failure analyses workflow is created during this project after unexpected behaviour occured. However, it has not been evaluated in other projects. It is valuable to evaluate the workflow for a new research setup during system engineering.

7.2.5 ViewCorrect Plotter setup

Due to lack of time some features of the ViewCorrect Plotter setup have not been implemented. At this moment, paper is fixed to the bottom plate. This should be replaced with some sort of clipper.

Besides a demonstration button, an emergency stop button and a brake stop button should be implemented. The demonstration button should be connected to a button connector of the MCB and a software process should detect a change of state and start a demonstration program stored in flash memory of the PC/104. The brake stop should also be connected to a button connector of the MCB. A predefined process in the FPGA configuration file should process this to a brake signal to the H-Bridge. The setup is still open, consequently dust and dirt can harm the construction. Besides it might be hazardous for bystanders, because the plotter can move with high velocity. It is recommended to make some sort of transparent cap.

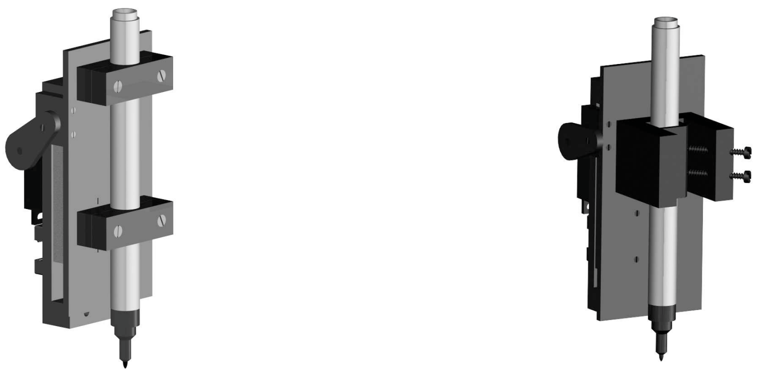
If higher accuracy is needed, it is recommended to redesign part of the gear of the x- and y-axis, because visual inspection shows still some non-linear behaviour. This was also visible in the validation of the model of the ViewCorrect Plotter setup.

A Realization of the recommendations for the ViewCorrect Plotter Setup

This appendix describes the design and realization of the recommendations for the ViewCorrect Plotter given after a previous project (Kuppeveld and Sprik, 2006). It starts with the pen mechanism (A.1), followed by the pen control (A.2), linear encoders (A.3) and the printed circuit board for the plotter I/O (A.4). Different options are discussed in these topics.

A.1 Pen mechanism

The pen mechanism has to be a stable construction, which claps the pen in its place. An easy way of replacing the pen or some other device is an issue. Another issue is to have the possibility to replace the pen or pen holder with another device like a (milling) cutter or inkjet head. Multiple options are available, only two creative options will be discussed. Figure A.1 shows both options. The designs are modelled in SolidWorks, which is a design tool for mechanical contructions.



(a) Pen mechanism option one (b) Pen mechanism option two

Figure A.1: Two types of pen mechanisms.

Both options clasp the pen at two points and the pen holder is removed easily, which is needed to be prepared for future projects. The first options needs multiple blocks with different diameters in order to clasp pens with different sizes. The second options only need to adjust the screws. Both options have the disadvantage that the place of the pen point differs with the size of the pen. This does not necessarily imply a problem, as long as the control software is aware of this. A construction with a fixed place of the pen point is rather difficult.

A.1.1 Conclusion

The second option is chosen, because of the easiness of replacing the pen.

A.2 Pen control

The servo motor (Hitec, 2007) requires a 3-5V DC peak to peak square wave pulse. The width of the pulse determines the angle of rotation of the motor axis. The motor axis drives the plastic handle, which is connected to the pen holder. The plastic handle moves the pen holder on the z-axis. The pulse duration is from 0.9 ms to 2.1 ms with 1.5 ms as center. The pulse frequency is 50 Hz.

In the previous design the pulse was generated from the plotter software. In order to generate a pulse with a width resolution of 0.1 ms, a clock frequency of 10 kHz in needed in the plotter software.

An option is to place a pulse generator in the FPGA of the Anything I/O board (Groothuis, 2004) of the PC/104 stacks. In this way the plotter software sends the necessary information to the Anything I/O board and the FPGA configuration sets the required pulse at a output pin. Consequently the plotter software can operate at a lower frequency.

The Anything I/O board has 72 general purpose I/O pins connected to an FPGA. The existing FPGA configuration at the CE laboratory offers a PWM generator. The disadvantage of this PWM generator is that it is only possible to generate a fixed PWM frequency for all PWM signals at 16.3 kHz. This frequency is produced from the PCI system clock (33 MHz). The duty cycle of the PWM signal is adjustable via a 11 bit register value.

This leaves two options for the FPGA configuration. The first option is to design a component which is specialized to control this type of servo motor and add this to the existing configuration. The second option is to adjust the existing PWM generator and make is possible to generate PWM signals at different frequencies.

This first option is easy to implement and requires less knowledge of the existing configuration. The second option is more complex, takes more time to design, but comes with a generic solution which offers more functionality to the existing FPGA configuration.

A.2.1 Conclusion

The second option is preferable, because now it is possible to re-use this functionality in future projects. Backward compatibility is an issue.

A.2.2 Design

In order to control the servo motor and the other motors at the ViewCorrect Plotter, it is necessary to have multiple PWM signals with different frequencies. The existing design consists of a flexible number of PWM generators and one PWM reference component. This has to be replaced by same number of PWM generators as PWM reference components, as depicted in Figure A.2. This makes it possible to have multiple PWM signals with different frequencies. The frequency output of the PWM reference components have to be adjustable via a register value.

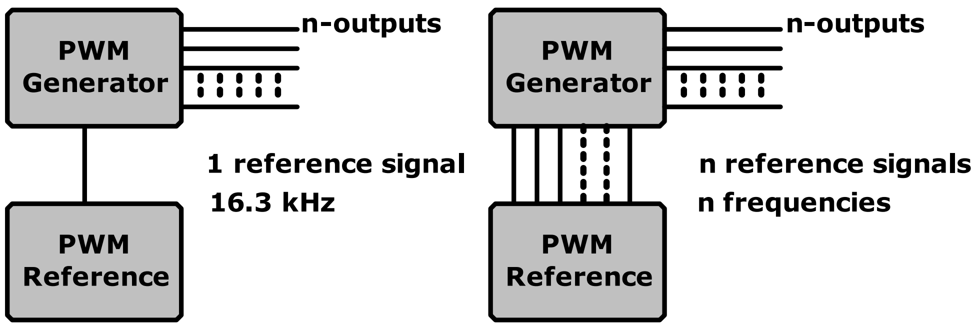


Figure A.2: PWM generator design.

In order to divide the output frequency a phase accumulator is used. The theory of a phase accumulator is explained in the next section.

A.2.3 Phase accumulator

A phase accumulator is frequently used part of a direct digital synthesis device. Direct digital synthesis is a method of producing an analog waveform, like a sine wave using a digital-to-analog conversion.

The phase accumulator produces the output frequency. The output frequency depends on two variables. The first is the reference clock frequency and the second is the binary number programmed into the frequency register.

The register value provides the main input to the phase accumulator. The phase is incremented each clock tick. The size of the phase increment determines the actual output frequency. In case of a change of value in the frequency register, the output frequency will change immediately. The binary width of the phase accumulator determines the minimum frequency, which is equal to the frequency step. Consequently more bits for the accumulator allows for a finer frequency tuning. The minimum frequency is defined by:

$$\Delta f = \frac{1}{2^n} * f_{in} \tag{A.1}$$

The output frequency is:

$$f_{out} = \frac{register\ value}{2^n} * f_{in} \tag{A.2}$$

N is the length of the phase accumulator in bits, f_{in} is the reference clock frequency, f_{out} is the output frequency. Δf is the frequency step. The jitter in the output signal depends on the jitter of the system clock. The system clock of the Anything I/O board is the 33 MHz PCI clock. The percentage jitter of the output signal will be reduced with frequency division, because the same amount of jitter occurs within a longer period.

A.2.4 Results

The frequency of the PWM signal is now adjustable by a 16 bit register. According equation A.1 and A.2 Δf and f_{out} are:

$$\Delta f = \frac{1}{2^{16}} * 16,3\ kHz = 0,25\ Hz \tag{A.3}$$

$$f_{out} = \frac{register\ value}{2^{16}} * f_{in}, \quad f_{in} \approx 16,3\ kHz \tag{A.4}$$

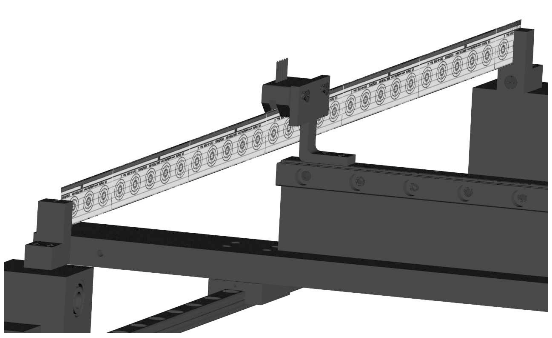
A decimal register value of 201 will produce a PWM output of 50 Hz. The configuration is backward compatible with the existing configuration (Groothuis, 2004). This is realized by initializing the register at an output frequency of 16.3 kHz. Appendix K describes how to program the FPGA configuration file manually.

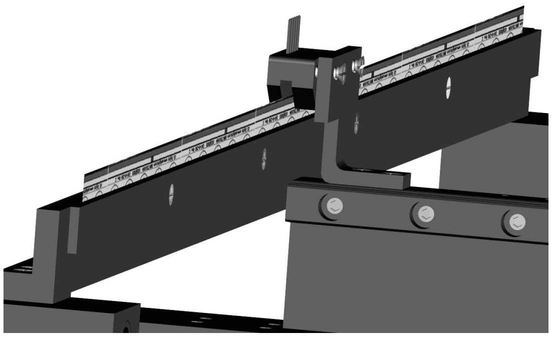
A.3 Linear encoders

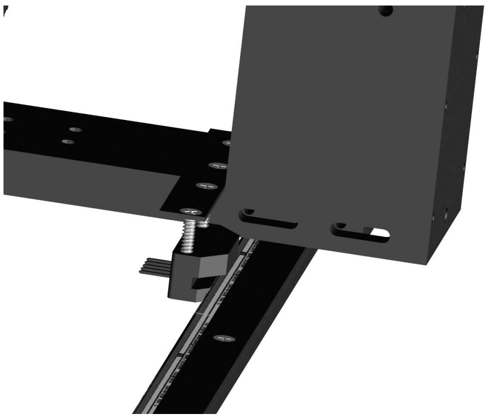
The exact position of the pen is measured by linear encoders. These encoders are an addition to the encoders placed on the motors. Both axes use an encoder. The x-axis uses two encoders.

Four feasible construction options are investigated for the x-axis. The first, second and the third option use a US Digital HEDS encoder (US-Digital, 2006) with a linear strips. This is the same encoder as used in the Mechatronic demonstrator (Dirne, 2005). The linear strip is made of thin plastic and therefore sensitive. The third option uses a Heidenhain metal plated encoder (Heidenhain, 2006).

All options have the linear read unit attached to the moving y-axis. The first option is to tighten the linear strips between both sides of the x-axis. The linear strip is unprotected. This is improved in the second option. The second option is to attach the linear strips to a L size framework which is mounted on both sides of the x-axis. The third option is to attach the linear strips between two metal strips and mount this to the bottom side of the plotter. The fourth option is to use a linear rail instead of a strip. This rail will be mounted on the bottom side of the plotter and the read unit will be attached to the y-axis. All options are depicted in A.3.


(a) Linear encoder option one


(b) Linear encoder option two


(c) Linear encoder option three

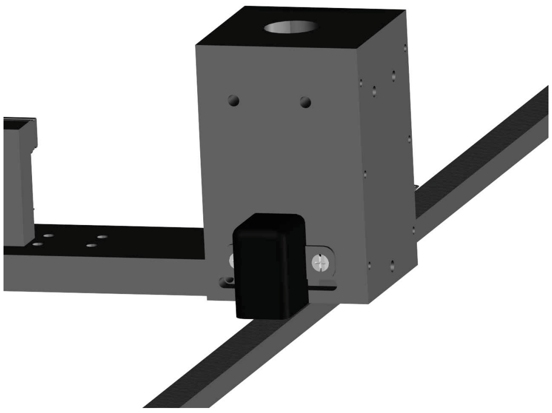

(d) Linear encoder option four

Figure A.3: Four types of linear encoder constructions.

The US Digital HEDS encoder and strip costs : €27 + €39. The accuracy of the US Digital set is 0.0705 mm. The Heidenhain encoder and read unit costs : €221 + €321. The accuracy of the Heidenhain set is 0.015 mm. Table A.1 shows the different options evaluated on price, construction time, accuracy of the encoder and robustness of the construction.

	Price	Construction time	Accuracy	Robustness
1	+	+	+	-
2	+	+	+	+/-
3	+	++	+	+
4	−	++	++	++

Table A.1: Comparison four options linear encoder x-axis.

The y-axis has due to size limitations only two feasible options. The first option is to attach the linear strip between metal strips, which are mounted on the same frame as the guidance rails. The read unit will be mounted on the pen mechanism. The second option is to use a linear rail on top of the strips.

A.3.1 Conclusion

The x-axis has four options. The first option is fragile, because the linear strip is unprotected. This is a disadvantage when a user is replacing the paper and by accident is breaking down the strip. The second option has a more robust framework, but still is awkwardly when replacing paper. The third and fourth do not have this disadvantage. The higher accuracy of option four is not needed in this project and for future projects it is possible to order linear strips with a higher accuracy. Consequently option three is implemented on the plotter.

The y-axis has two options. The first option is chosen, because it is not necessary to have such a high accuracy as in option two. In Figure A.4 the ViewCorrect Plotter is depicted with the linear encoders and the new pen mechanism.

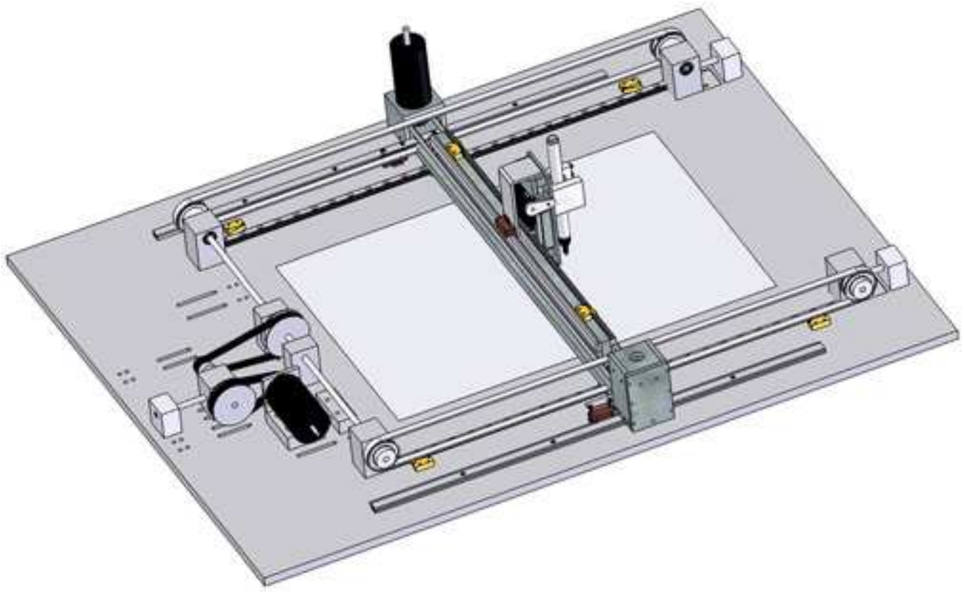


Figure A.4: Isometric view of the ViewCorrect Plotter with linear encoders and the new pen mechanism.

A.4 Printed circuit board for the plotter I/O

In this section the overview, requirements and the design of the printed circuit board are discussed. It ends with the resulting board.

A.4.1 Overview

The PCB is connecting the I/O signals of the Anything I/O board to the different electronic parts of the ViewCorrect Plotter setup. Consequently next to a connection interface, peripheral electronic is needed to drive the encoders etc. The PCB is named as Motor Control Block.

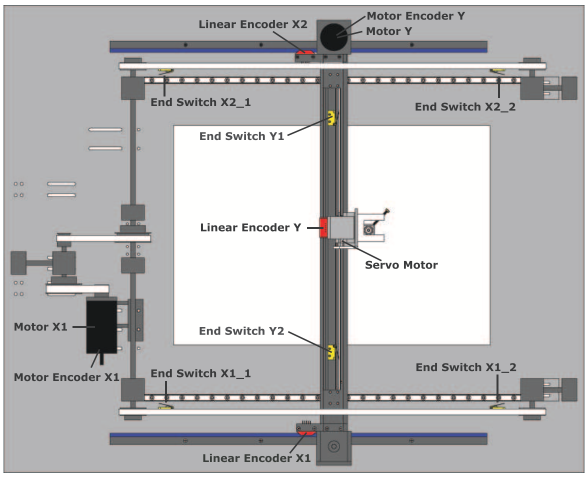
One of the requirements set to this setup was preparing for distributed control. Every motor, except for the servo-motor in the pen mechanism, will be controlled by a controller node in that case. The controller node is implemented by a ECS. It is not required that one controller node, can be replaced by another controller node in case of a failure. Consequently, it is not needed to switch signals back and forth to the different controller nodes. All three controller nodes are connected to a supervisor by a fieldbus. This MSc-project will not implement distributed control. Another requirement for this setup was a brake and a demonstration mode button.

The Anything I/O board has three connectors with 24 I/O pins each. The ViewCorrect Plotter has two motors each for every axis, but is prepared to drive the x-axis with two motors. This makes distributed control for three motors and the peripheral electronics. Each motor is driven by the H-Bridge PCB which needs 4 I/O. The motor encoder needs 2 I/O, pen control needs 2 I/O, an end switch needs 1 I/O, the linear encoder needs 2 I/O and the brake and demonstration mode buttons need 2 I/O. Table A.2 shows an overview of the I/O signals. Figure A.5(a) shows a schematic topview of the ViewCorrect Plotter with named components.

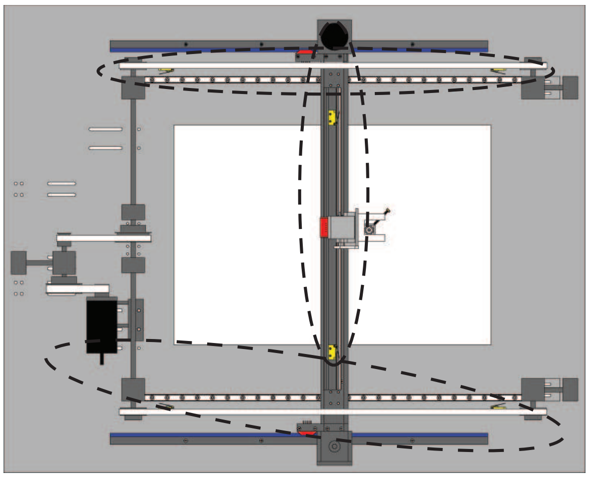
Part	**Number needed**	**I/O each**	**I/O total**
H-Bridge	3	4	12
Motor encoder	3	2	6
Linear encoder	3	2	6
End switch	6	1	6
Buttons	2	1	2
Pen control	1	2	2
		Total I/O	34

Table A.2: Overview I/O signals.

The ViewCorrect Plotter setup is divided into three separate control parts. Each part consist of a motor, a motor encoder, a linear encoder and two end switches, which are mounted near the guidance rail driven by the motor. These components are combined in one control part, because they are (directly) related to each other. Figure A.5(b) shows the three control parts.



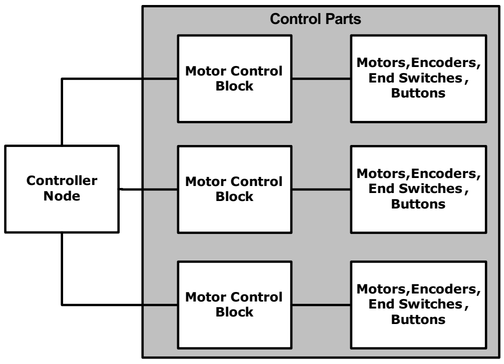
(a) Schematic topview with named components



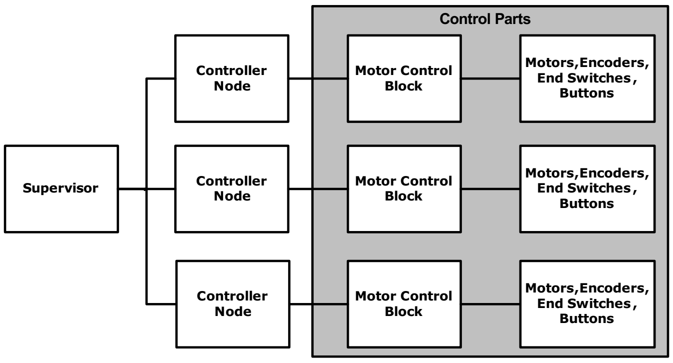
(b) Three control parts

Figure A.5: Schematic topview ViewCorrect Plotter setup.

The buttons and pen control are assigned to a random control part. The controller node controls his part. The PCB should be prepared for distributed control and single control. Figure A.6(a) shows an overview of single control. In Figure A.6(b) a schematic overview of distributed control is depicted.


Control Parts
Controller Node
Motor Control Block
Motor Control Block
Motor Control Block
Motors,Encoders, End Switches, Buttons
Motors,Encoders, End Switches, Buttons
Motors,Encoders, End Switches, Buttons

(a) Overview of the electronics with one controller node



(b) Overview of the electronics with distributed control

Figure A.6: Two types of controller systems.

A.4.2 Requirements

The requirements for the PCB are:

- The plotter is a research setup for real-time embedded software. Consequently extra delay in the software caused by the electronics of the PCB is undesired.
- The PCB is connected to the Anything I/O board. At CE laboratory two versions of this board are available. The difference concerning this project is the operating voltage. The 4165 board has an adjustable voltage of either 3,3 or 5V. The 4168 board has an

adjustable voltage of either 1,8 or 3,3V. This makes 3,3V the preferred operating voltage of the PCB. Making the PCB 5V compatible, prevents damage from human faults by setting the jumper at the 4165 board wrong.

- To prevent faulty connections, female and male connectors which have only one option of connecting should be chosen.
- LED's have to implemented for debugging and demonstration purposes. They can indicate the status of I/O signals and this will help debugging.
- The Anything I/O board connector has to be implemented twice and connected to each other. This is a helpful feature for signal measuring or debugging. For future use, additional electronics can be connected to the PCB easily.
- The encoders and pen control require a operating voltage of 5V DC. Therefore the PCB should be equipped with 5V power supply.
- I/O signals which are used to control (servo) motors should be in a safe state when the controller nodes are not running, but connected to a power supply. In that case the I/O signals are in tri-state. Pull up resistors or other electronics will prevent damage or undesired behaviour.
- For future use it is useful to have two type of encoders connections, differential and non-differential.
- The PCB will be used with the H-Bridge in order to control a DC motor. If possible, it is practical to make the PCB stackable with the H-Bridge.

A.4.3 Design

The electronics of the buttons and the pen control will implemented on only one PCB. However the circuit is available on every PCB. In this way it is possible to use three identical circuit boards, which is an advantage for the costs.

Part	**I/O each**	**I/O total**
H-Bridge	4	4
Motor encoder	2	2
Linear encoder	2	2
End switch(2)	1	2
Buttons	1	0 or 2
Pen control	2	0 or 2
	Total I/O	10 or 14

Table A.3: Overview I/O signals of one PCB.

Table A.3 shows a total of 10 or 14 I/O signals on each PCB. This leaves 14 or 10 I/O free and can be used for future projects. This can be connected to the second connector.

A.4.4 Results

A photo of the designed PCB can be seen in Figure A.7. In Appendix G a user manual can be found. The schematics and the pin numbering of the board be found in Appendix H.

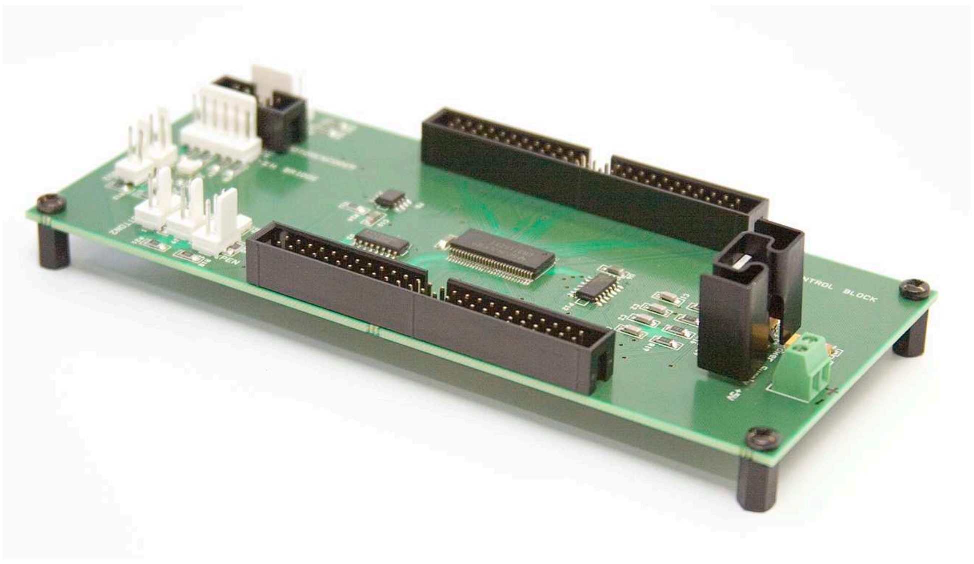


Figure A.7: Motor Control Block - Cornelis Kooistra CE 2007.

B Model of the ViewCorrect Plotter Setup

This appendix depicts the model of the ViewCorrect Plotter setup (Figure B.1), which describes the behaviour of the setup. The parameters of this model are depicted in Table B.1. Simulation plots of the validation of the model and controller design are depicted in B.2.

B.1 Model and parameters

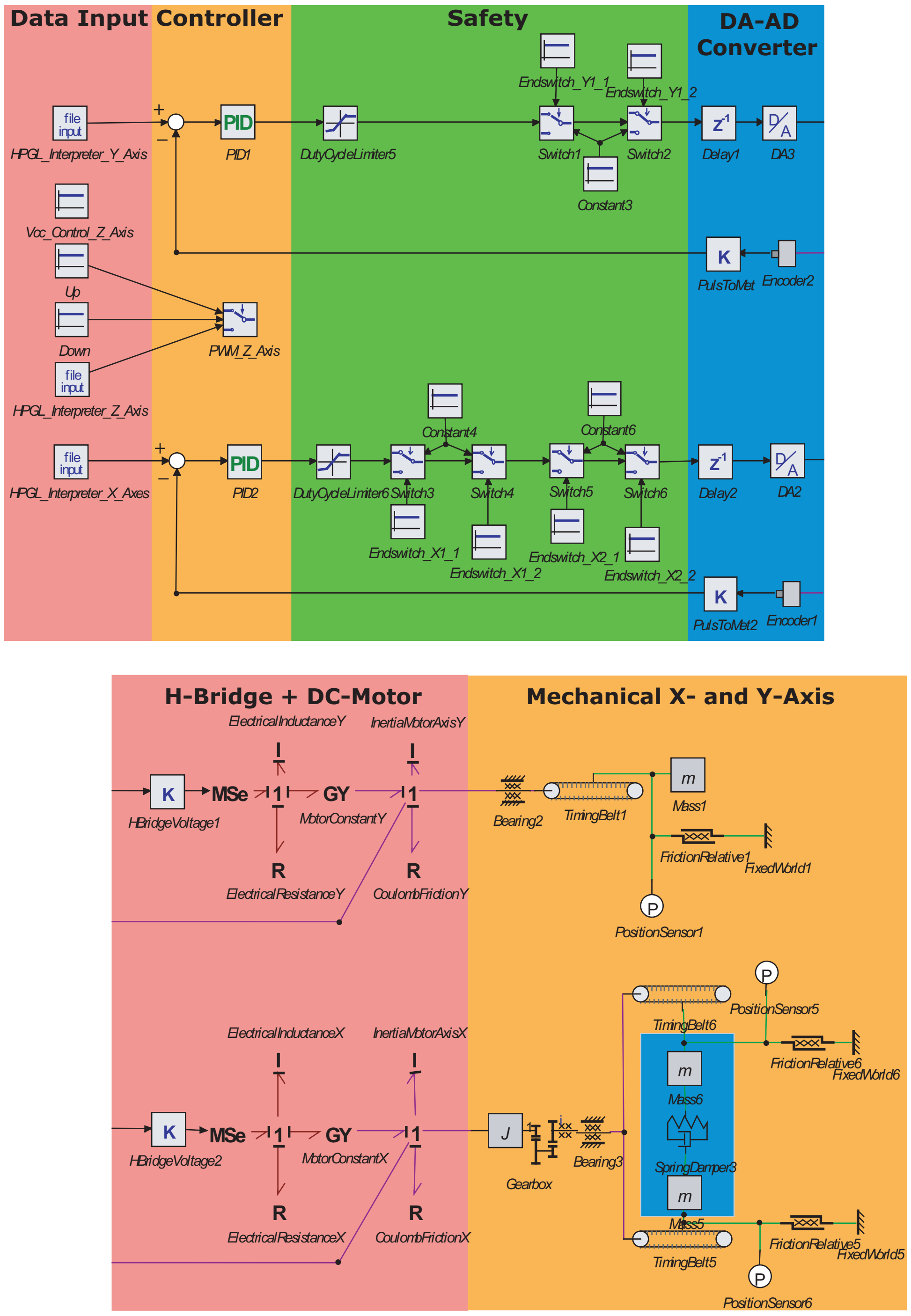


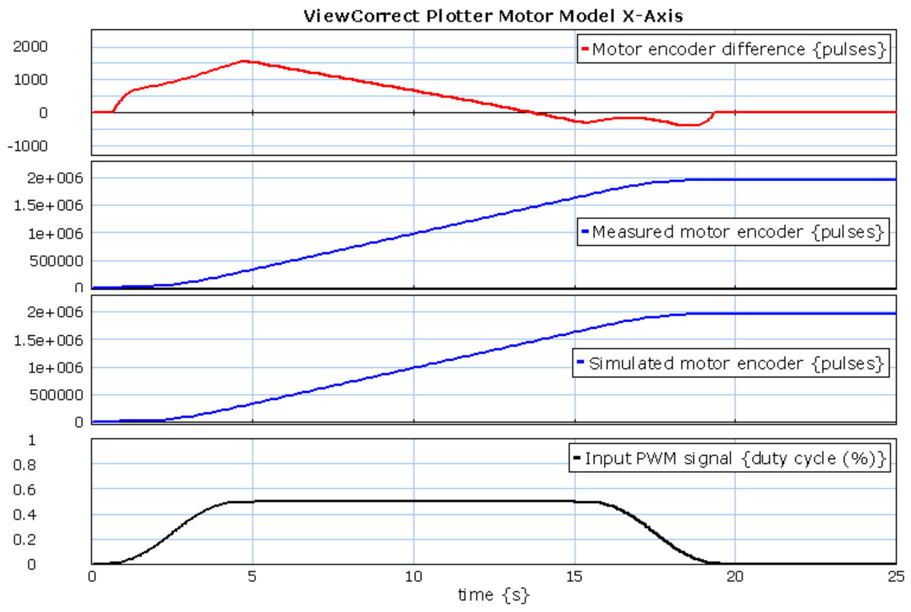
Figure B.1: Model of the ViewCorrect Plotter setup.

Name	**Description**	**Value**	**Unit**
CoulombFrictionX		0.0041374	N
ElectricalInductanceX		0.0823m	H
ElectricalResistanceX		0.317	Ω
HBridgeVoltage1,2		24.4	V
InertiaMotorAxisX		$1.38 \cdot 10^{-5}$	kgcm^2/rad
MotorConstantX		30.2m	Nm/A
CoulombFrictionY		0.003822	N
ElectricalInductanceY		0.201m	H
ElectricalResistanceY		1.11	Ω
InertiaMotorAxisY		$6.99 \cdot 10^{-6}$	kgcm^2/rad
MotorConstantY		36.4m	Nm/A
DutyCycleLimiter5	Maximum	±0.2	
DutyCycleLimiter6	Maximum	±0.9	
PID1	Kp	40	
	τ_i	$21.5 \cdot 10^{-3}$	
	τ_d	$5.38 \cdot 10^{-3}$	
PID2	Kp	220	
	τ_i	$8.19 \cdot 10^{-3}$	
	τ_d	$2.04 \cdot 10^{-3}$	
DA2,DA3	Bits	12	
Encoder1,Encoder2	Pulses	2000	/rev
Bearing3	Rotational friction	35m	Nms/rad
FrictionRelative5,6	Coulomb friction	1	N
FrictionRelative1,5,6	Viscous damping around v=0	1M	Ns/m
FrictionRelative1,5,6	Viscous damping elsewhere	6	Ns/m
FrictionRelative1,5,6	Friction coefficient	0.0001	s/m
Gearbox	Ratio	0.1162	
	Inertia	$1.23852 \cdot 10^{-5}$	kgm^2/rad
Mass5,Mass6	Weight	1.2	kg
SpringDamper3	Spring constant	10k	N/m
	Damping	1k	Ns/m
TimingBelt1	Pulley radius	11.625m	m
	Belt area	0.004	m^2
	Belt length	0.8	m
TimingBelt1,5,6	Damping	1	Ns/m
TimingBelt1,5,6	Elasticity	1M	N/m^2
Bearing2	Rotational friction	1m	Nms/rad
FrictionRelative1	Coulomb friction	600m	N
Mass1	Weight	0.3	kg
TimingBelt5,6	Pulley radius	15.65m	m
TimingBelt5,6	Belt area	0.005	m^2
TimingBelt5,6	Belt length	1	m

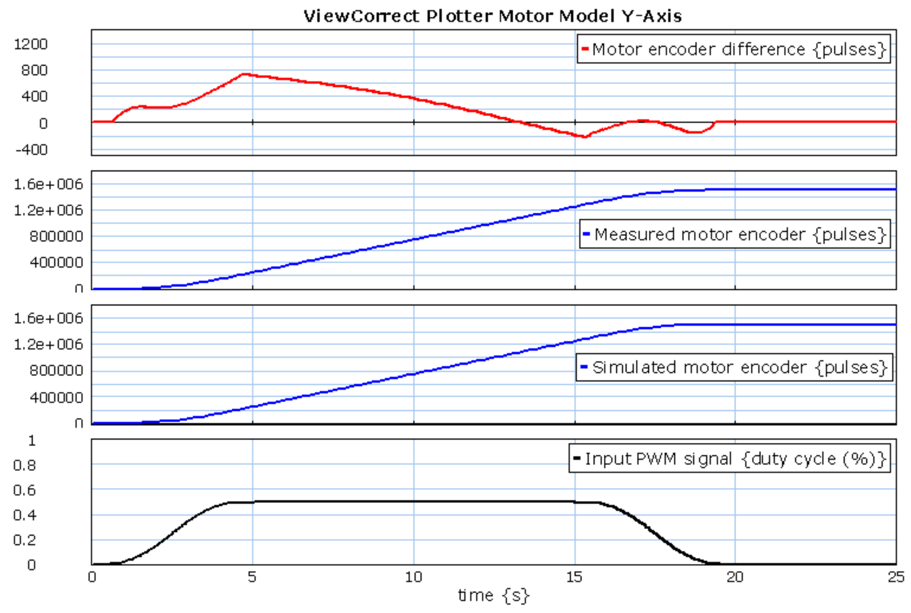
Table B.1: Parameters model of the ViewCorrect Plotter setup.

B.2 Model validation and controller design

These two figures (B.2) show the validation of the submodels *data input* and *DA and AD conversion* and the motor part of the *plant model*. The motor part of the *plant model* is the H-bridge and the DC-motor. The submodels *controllers* and *safety* are not part of this validation. The difference is at the most 0.5 percent for the x-axis and 0.3 percent for the y-axis.



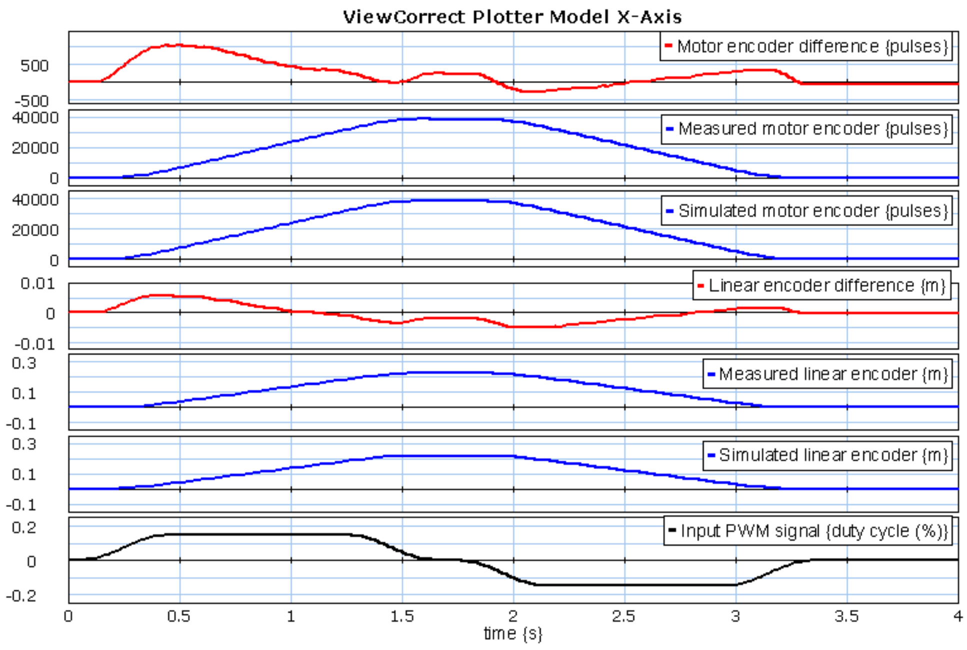
(a) X-Axis



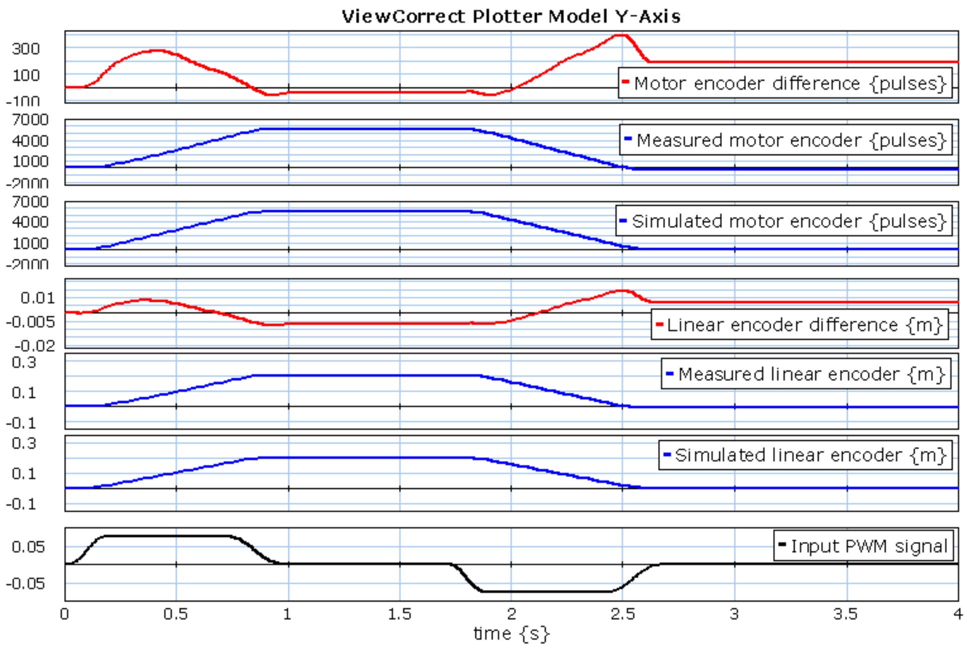
(b) Y-Axis

Figure B.2: Validation motor model of the ViewCorrect Plotter setup

These two figures (B.2) show the validation of the the *plant model*, *data input* and *DA and AD conversion*. The submodels *controllers* and *safety* are not part of this validation. The difference is at the most five percent for both axes.



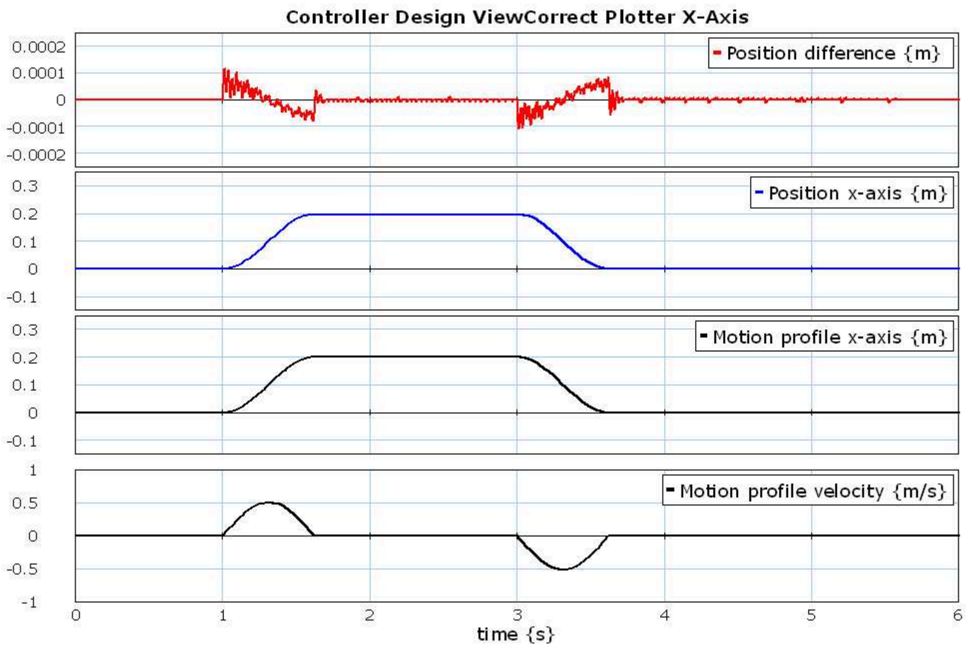
(a) X-Axis



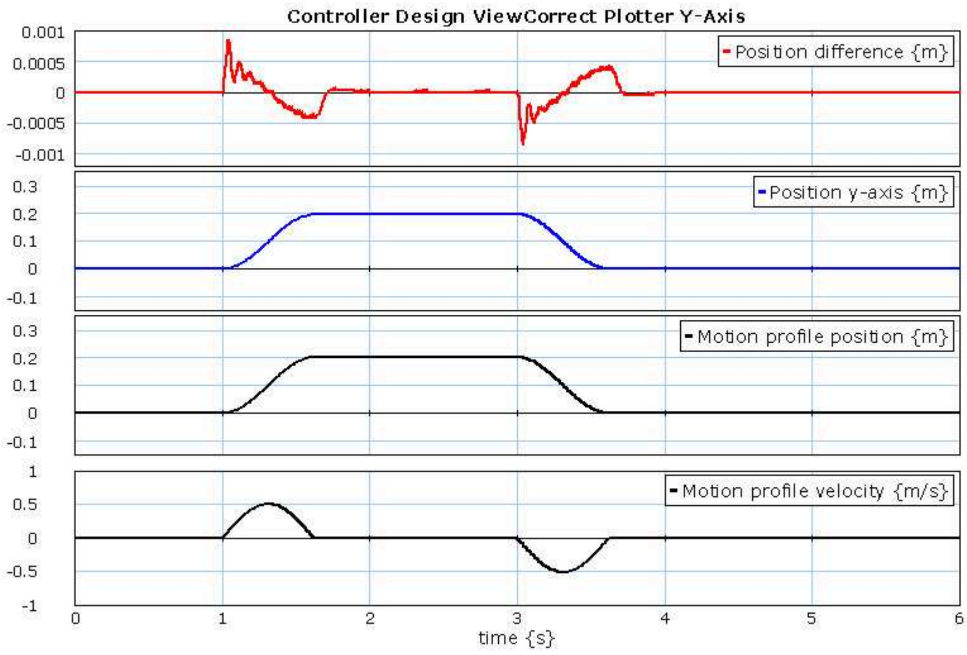
(b) Y-Axis

Figure B.3: Validation part of model of the ViewCorrect Plotter setup

These two figures (B.4) show the designed controllers for the x- and y-axis and their responses on a motion profile. The accuracy is less than 1 mm during a motion and less than 0.1 mm when the motion is finished for both controllers.



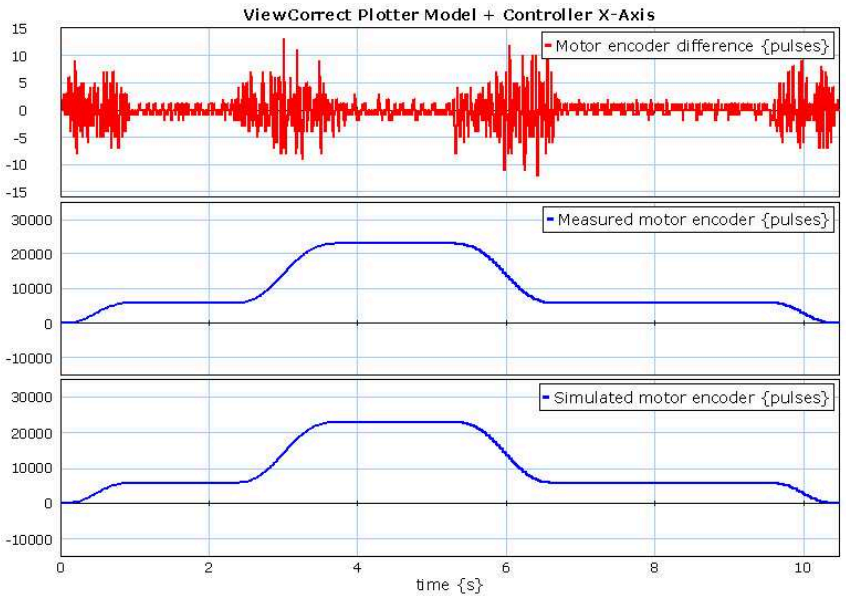
(a) X-Axis



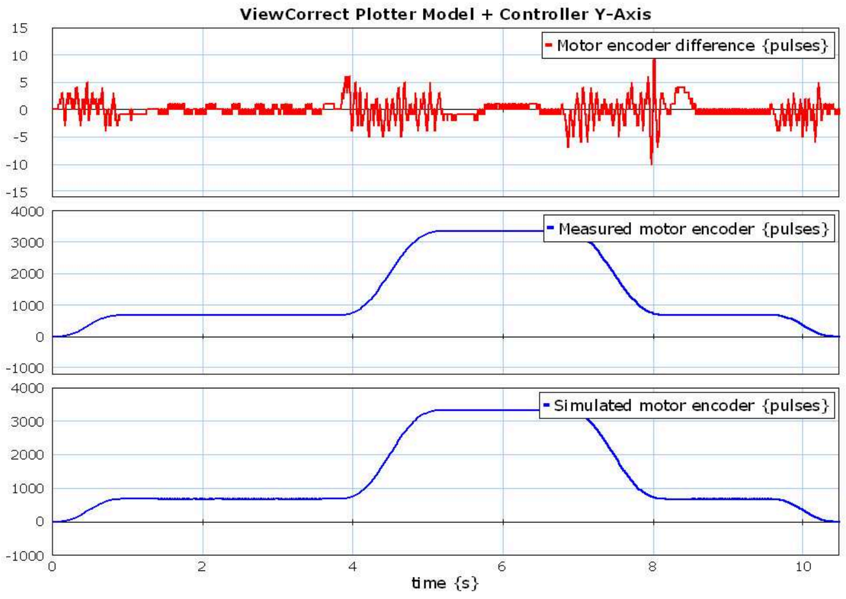
(b) Y-Axis

Figure B.4: Controller design ViewCorrect Plotter controllers

These two figures (B.5) show the validation of the model of the ViewCorrect Plotter setup as depicted in B.1. The accuracy of the model is 0.03 mm during a motion with a maximum velocity of 0.2 m/s and 0.005 mm (1 pulse) when the motion is finished for the x-axis. For the y-axis, these parameters are 0.1 and 0,04 mm (1 pulse). The error between simulated and measured encoder pulses is periodical during a motion. It looks like the dynamics are caused by the gear construction, because the number of periods is a multiple of the number of rotations of the motor axes.



(a) X-Axis



(b) Y-Axis

Figure B.5: Validation model of the ViewCorrect Plotter setup

C Vector Drawing Formats

This appendix contains a table with the most common vector drawing formats.

Extension	**Description**
3DS	3D Studio
906	Calcomp plotter
AI	Adobe Illustrator
CAL	CALS subset of CGM
CDR	CorelDRAW
CGM	Computer Graphics Metafile
CH3	Harvard Graphics chart
CLP	Windows clipboard
CMX	Corel Metafile Exchange
DGN	Intergraph drawing format
DMPL	Houston Instruments plotter language
DSF	Micrografx Designer 6.x
DXF	AutoCAD
DWG	AutoCAD
EMF	Enhanced metafile
EPS	Encapsulated PostScript
ESI	Esri plot file (GIS mapping)
FMV	FrameMaker
GBR	Gerber PCB format
GCA	IBM GOCA
G4	GTX RasterCAD - scanned images into vectors for AutoCAD
HP-GL(2)	HP graphics language
IGF	Inset Systems (HiJaak)
MCS	MathCAD
MET	OS/2 metafile
MRK	Informative Graphics markup file
P10	Tektronix plotter (PLOT10)
PCL	HP LaserJet
PCT	Macintosh PICT drawings
PDW	HiJaak
PIX	Inset Systems (HiJaak)
PS	PostScript different levels
RIS	AUCOTEC CAD format
RLC	Image Systems ”CAD Overlay ESP” vector files overlaid onto raster images
RTL	HP raster and vector graphics language
SSK	SmartSketch
SVG	Scalable vector graphics (XML)
WMF	Windows Metafile
WPG	WordPerfect graphics

Table C.1: Vextor graphics formats.

D Plot Files and Languages

This appendix gives some additional information about the plot file and the different plot file languages. The most common plot languages for plotters are HP-GL, HP-GL/2, HP-PCL, HP-RTL, DMPL or Gerber. In the following sections the plot file and plot file languages are described.

D.1 Plot files

Most of the plot files have a extension .plt. The extension .plt does not tell anything. It is a extension that is used to identify output files that are intended to be sent to a plotter. The extension .plt is given to a plot file from any driver by a program like AutoCAD. A .plt file means that it is some sort of plot file for some sort of plotter. In order to know what can be done with a .plt file, it has to be known where it came from and what is was made for.

A .plt file can contain some device specific code. Not every language compatible plotter understands these device specific code. Furthermore the capabilities of the devices may be different, for example an A3-size plot file send to an A4-size device will cause not preferable results. Summarized .plt files can be sent to the plotter they were generated for.

D.2 Hewlett Packard Graphics Language (HP-GL)

HP-GL is a vector graphics file format developed by Hewlett Packard. Originally it was intended to drive pen plotters. It was taken up by the plotter industry as a standard plotting language and migrated to cutting plotters when they appeared. Most commands are vector oriented; pick up the pen, put down the pen, move from here to there. Very few complex commands, like character plot instructions, are part of HP-GL since the original plotters did not include powerful Central Processing Units (CPU). HP-GL commands are two letter codes that represent the function of the command, for example IN for initialize. After the two letter mnemonic, there may be one or more parameters that identify details of how to process the command. Two versions of HP-GL exist, which differ in coordinate systems. Small-format plotters, including A- and B-ISO paper size plotters, locate the origin at the lower-left corner; large-format plotters, including D- and E-ISO paper size plotters, locate the origin at the center of the media.

D.3 Hewlett Packard Graphics Language 2 (HP-GL/2)

HP-GL/2 was develop as a successor to HP-GL. HP-GL/2 was created in 1988 and supports more advanced features like grey shades for lines, screened lines, pen width settings and long axis plotting. HP-GL/2 also includes quite a lot of raster controls and commands since most modern plotters are actually raster plotters instead of pen plotters.

D.4 Digital Microprocessor Plotter Language (DMPL)

DMPL is a vector graphics file format from Houston Instruments that was developed for their pen plotters and later used on their cutting plotters. DMPL and HPGL are nearly identical languages. In this way it is possible to convert files easily from one language to the other or merge plot files.

D.5 Hewlett Packard Printer Command Language (HP-PCL)

HP created PCL to provide an efficient way to control printer features across many different printing devices. PCL was created in the late 1970s. Along the years different version were developed. PCL versions differ in functionality (for example font type support: bitmap fonts, scalable fonts, raster graphic compression methods, HP-GL/2 graphic support). The current version is PCL XL. PCL commands are compact escape sequence codes that are embedded in the print job before being sent to the printer. HP-PCL formatters and fonts are designed to quickly translate application output into high-quality, device-specific, raster print images. PCL is the most widely spread printer language in the laser printer market today. There are six major levels of PCL. The creation of these levels was driven by the combination of printer technology developments, changing user needs and application software improvements.

The PCL printer commands activate the printer features. HP provided four general types of HP printer language commands. Control codes, PCL commands, HP-GL/2 commands and PJL commands.

A control code is a character that initiates a printer function (for example, Carriage Return (CR), Line Feed (LF), Form Feed (FF), etc.).

PCL commands provide access to the printer's PCL control structure. The PCL structure controls all of the printer's features except those used for vector graphics, which are controlled by the HP-GL/2 commands. This design provided easy use from high level programming languages and in reality, made the PCL the industry standard. The terms are used interchangeably. Once a PCL command sets a feature of the printer that feature remains set until that PCL command is repeated with a new value, or the printer is reset to default. In other words a feature is turned on and then turned off.

D.6 Hewlett Packard Raster Transfer Language (HP-RTL)

This is a raster graphics language based on the HP-PCL language. Many of its commands are the same as those of HP-PCL. It supports colour plotting with customer-defined palettes, combined vector and raster plotting, plot scaling and clipping and several data compression methods. HP-RTL is only available on devices that support HP-GL/2 since it interacts with HP-GL/2.

D.7 Gerber

A Gerber file has its own file extension like .gbr or .gbx. A Gerber file is a standard format used by PCB manufactures and contains necessary information to draw circuit boards, like signal traces, drilled holes, milling and cutting information. The Gerber file is named after the Gerber Scientific Instruments Company, a manufacturer of for example photo plotters. These files are produced by specialized Electronic Design Automation (EDA) software like OrCAD or Eagle. A Gerber file consists of X,Y co-ordinates by commands that where the PCB drawing starts, the shape and where it ends. In addition to the co-ordinates it contains data information the shapes and sizes of lines, holes or other features. The current most used data format of Gerber is the Extended Gerber format, RS-274X.

E Workflow Output Plotter

This appendix depicts screenshots of the tools used in the workflow described in in subsection 4.1.1 to illustrate this workflow. This workflow allows the user to make a drawing in a CAD drawing package and plot this drawing with the plotter It shows a screenshot of CorelDraw (Figure E.1), which is a CAD drawing package, a screenshot of HPGLview (Figure E.2) to show a graphical representation of the plot file and a screenshot of a 20-Sim simulation plot (Figure E.3) to show the movements of the plotter according the setpoints generated by the drawing to motion translator.

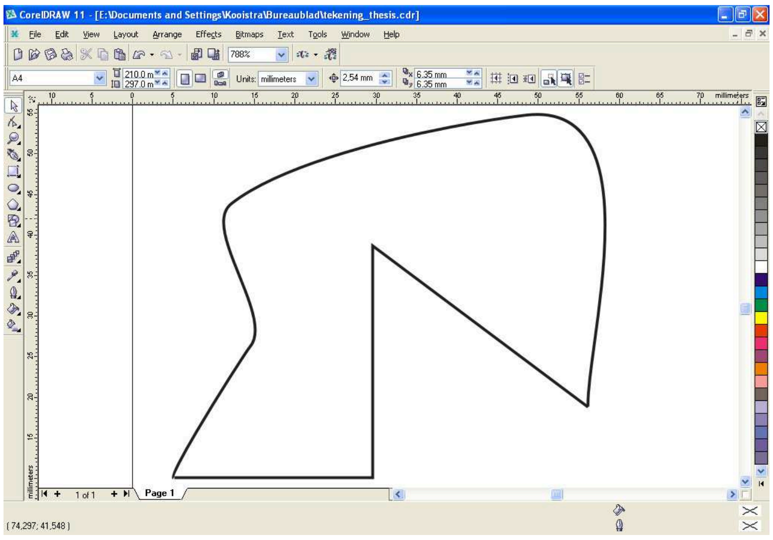

CorelDRAW 11 - [E:\Documents and Settings\Kooistra\Bureaublad\tekening_thesis.cdr]
File Edit View Layout Arrange Effects Bitmaps Text Tools Window Help
788%
A4
210.0 mm
297.0 mm
Units: millimeters
2.54 mm
6.35 mm
6.35 mm
millimeters
1 of 1
Page 1
(74,297; 41,548)

Figure E.1: Screenshot CAD drawing package: CorelDraw.

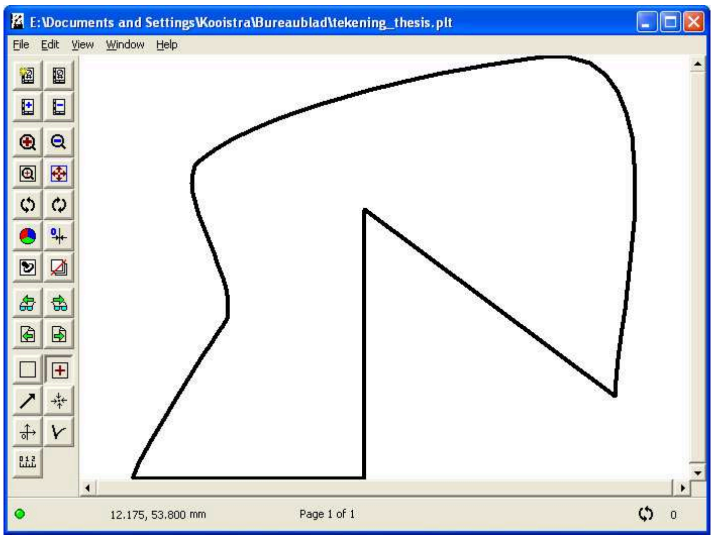

E:\Documents and Settings\Kooistra\Bureaublad\tekening_thesis.plt
File Edit View Window Help
12.175, 53.800 mm
Page 1 of 1
0

Figure E.2: Screenshot graphical representation of the plot file: HPGLview.

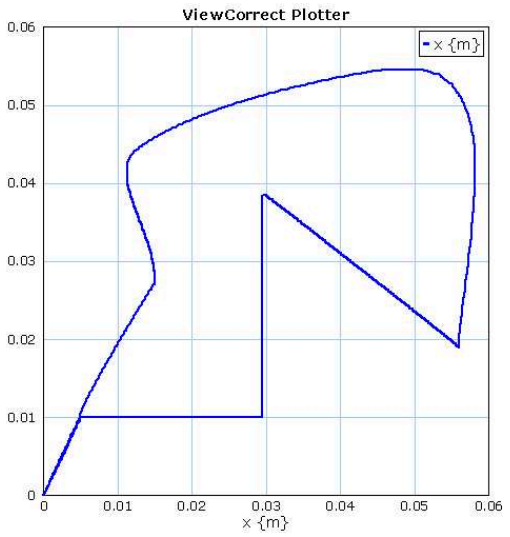


Figure E.3: Screenshot simulated movement of the plotter: 20-Sim.

In Figure E.3 an extra line is depicted from the origin to the drawing. This is because of the fact that it shows the movements of the pen, which starts in the origin and returns to the origin. Another reason is that in 20-Sim it is not possible to turn a plot line on or off during a simulation. Consequently all the movements are plotted or none. It is not possible to turn it on only when the pen is down.

F Plotter Software in gCSP

This appendix describes the plotter software written in gCSP of section 4.5 in more detail.

F.1 Data input

The process called *DataInput* is divided into two processes as can be seen in Figure F.1. *EncoderFeedback* reads the encoder value from the Anything I/O board and process *FileInterpreter* calculates new reference values for the controller according the drawing plot file.

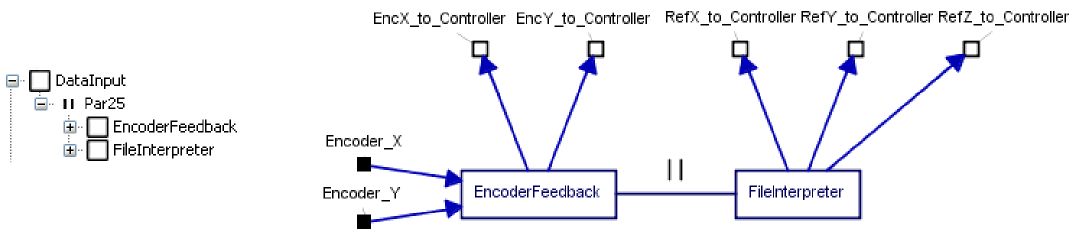


Figure F.1: DataInput.

Figure F.2 shows the details of *EncoderFeedback*. The reader reads the values from the link driver and in the code block these values are converted to meters. In Figure F.3 process *FileInterpreter* is depicted. In the code block the reference values for all three axes are calculated and written to the relevant variables.

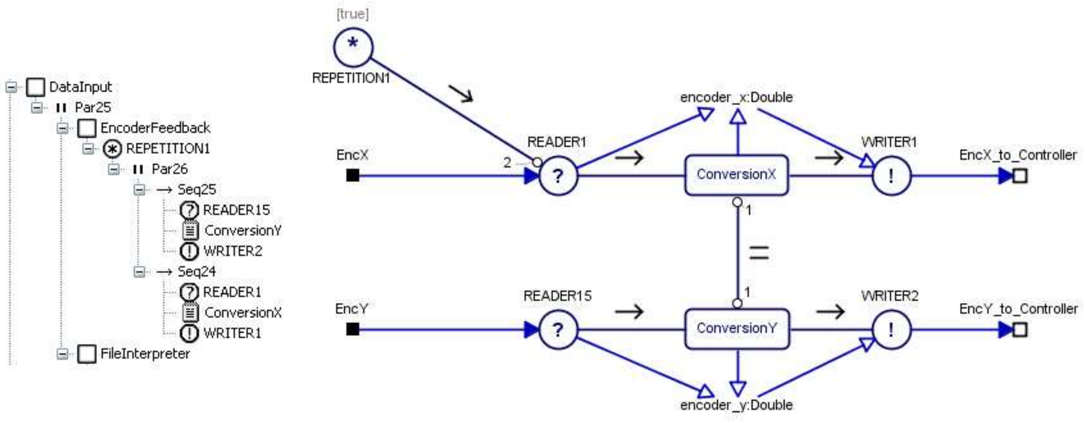


Figure F.2: EncoderFeedback.

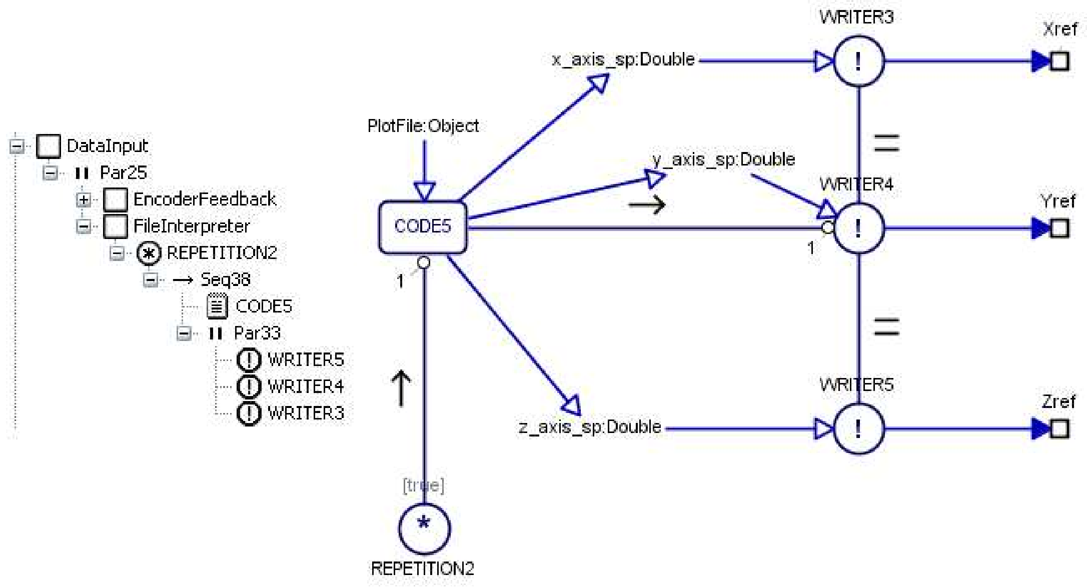


Figure F.3: FileInterpreter.

F.2 Controller

In process *Controller* new steering values are calculated with controller designed in 20-Sim (Block Controller_X_Y). These values are applied as PWM signals to the x- and y-axis. The PWM signal for the z-axis has only two values, one for the up and one for the down position of the pen. This can be derived from the reference value of the z-axis. The fourth signal is the control signal of the servo motor. This signal activates the power supply of the servo motor. The power supply is activated during run time of the software. Figure F.4 shows details of process *Controller*.

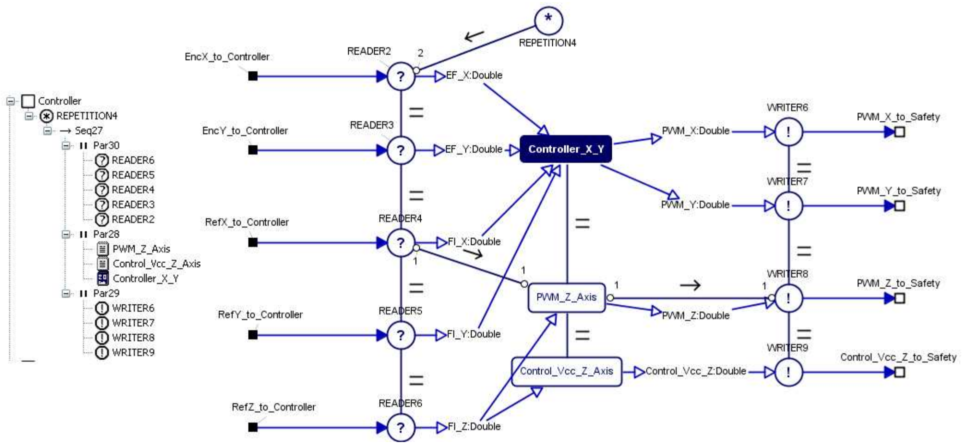


Figure F.4: Controller.

F.3 Safety

The process called *Safety*, depicted in Figure F.5, limits the PWM signals of the x- and y-axis to a safe domain if necessary. Besides the PWM signals are set to zero in case an end switch of the corresponding axis is hit. The values of the end switches are read by the link drivers.

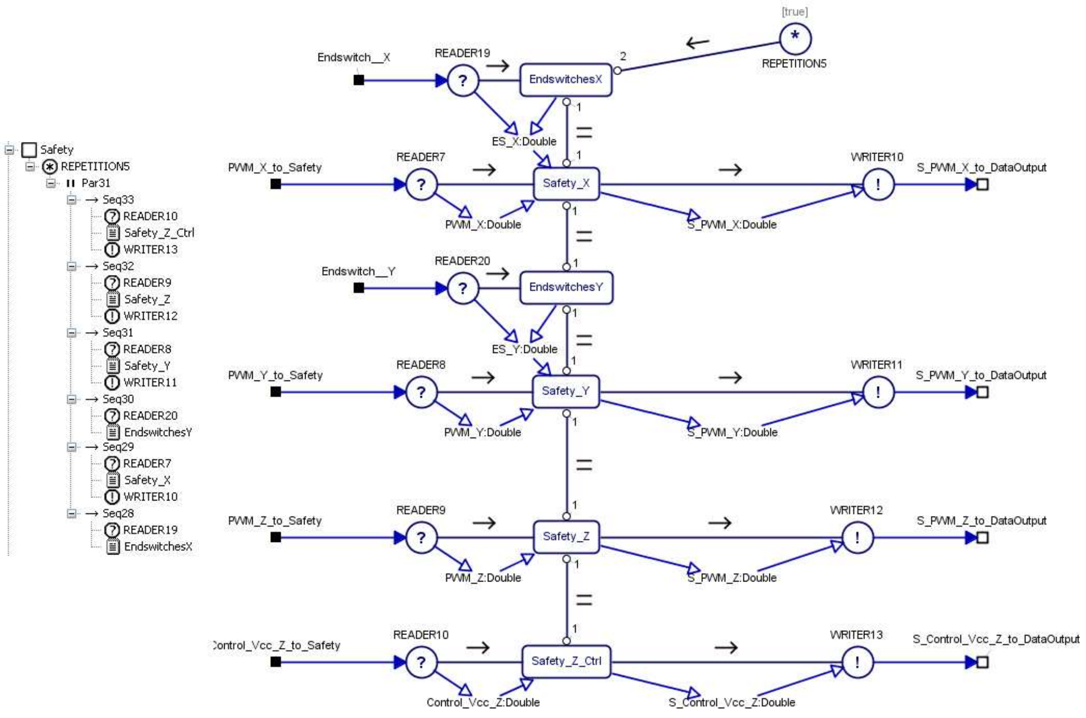


Figure F.5: Safety.

F.4 Data output

In process *DataOutput* all signals: PWM for the x-,y- and z-axis and a control signal for the power supply of servo motor are written to the Anything I/O board. Process *DataOutput* is depicted in Figure F.6.

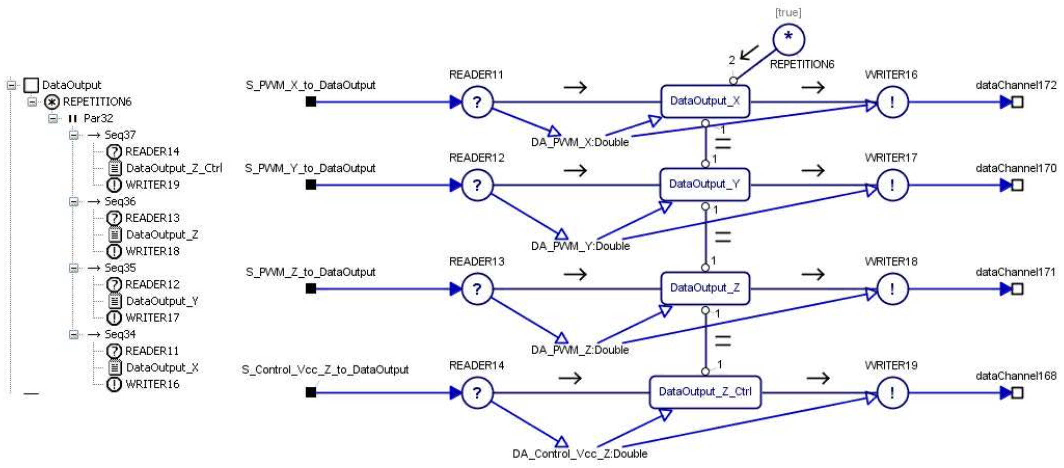


Figure F.6: DataOutput.

G Motor Control Block User Manual

This appendix describes all features and connections of the Motor Control Block (MCB). The MCB has been developed during the ViewCorrect Plotter project. Therefore the MCB contains some components specially for this project. The MCB has the same form factor and is stackable with the existing CE H-bridge PCB (van den Berg, 2006) to a complete electronic circuit to control a DC motor with an I/O board. Figure G.1 shows a top view of the board.

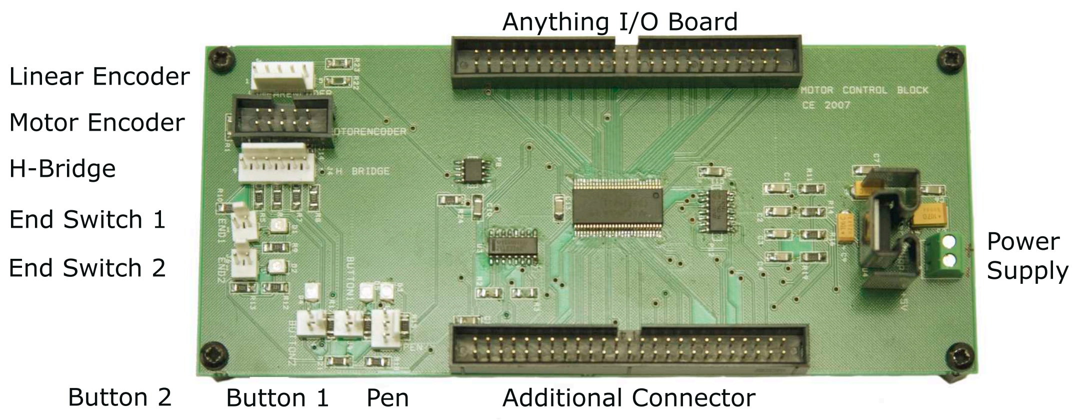


Figure G.1: Motor Control Block - Cornelis Kooistra CE 2007.

G.1 Features

- Bidirectional 5V or 3.3V logic compatible.
- Non-differential encoder connection.
- Differential encoder connection.
- Additional I/O connector for measuring or extension with additional electronics.
- LED's for indicating control signals or safe operating areas.
- 4 Connections for normally closed end switches or buttons.
- 5V DC power supply.

G.2 Connections

Power Supply

A 5V DC is required to feed the MCB and the electronics connected to the board. In normal operation a 5V - 1A power supply is sufficient.

Linear Encoder

This connector is made for a non-differential encoder, like the HEDS 9200.

Motor Encoder

This connector is made for a differential encoder, like the HEDS 9200 with integrated line driver.

End Switches

These pins can be connected to normally closed end switches. A closed connection is indicated by a green LED. This means a safe operating area. If an end switch is reached, the FPGA configuration will generate a brake signal to the H-Bridge.

Buttons

These pins can be connected to normally closed buttons in order to activate a demonstrating program or as a emergency brake stop. A closed connection is indicated by a yellow LED. This means a control signal. The buttons use the same electronic circuit as with the end switches. Consequently they can be used for extra connections for end switches. Only the yellow LED should be substituted with a green LED.

Pen

This connector is made for the servo motor, which is used in the ViewCorrect Plotter setup. The 5V DC power supply has to be activated with a low Pen_Vcc_Control signal. A yellow LED indicates a active power supply. This supply is limited to 500mA.

Anything I/O Board Connector

These pins have to be connected to one of the three connectors of the 4165 or 4168 Anything I/O board.

Additional Connector

This connector can be used for measuring the signals available at the Anything I/O board connector which can be useful for testing purposes. It can also be used to connect additional electronics to the free I/O ports.

G.3 Components

- Current limited distribution power switch, TPS2041AD, (Texas Instruments, 2000).
  The TPS2041AD is a current limited power switch with internal charge pumps to minimize current surges. This device limits the current to 500mA.
- Bus switch with 5V tolerant level shifter, SN74CB3T16211DLR, (Texas Instruments, 2005).
  The SN74CB3T16211 is a high-speed TTL-compatible signal operation on all data I/O ports. The SN74CB3T16211 supports systems using 5V TTL, 3.3V LVTTL, and 2.5V CMOS switching standards or user defined switching levels till 0.8V.
- Quadruple differential line receiver, AM26LS32A, (Texas Instruments, 2002).
  The AM26LS32A is a quadruple differential line receiver and translates differentials signals to non-differential signals.

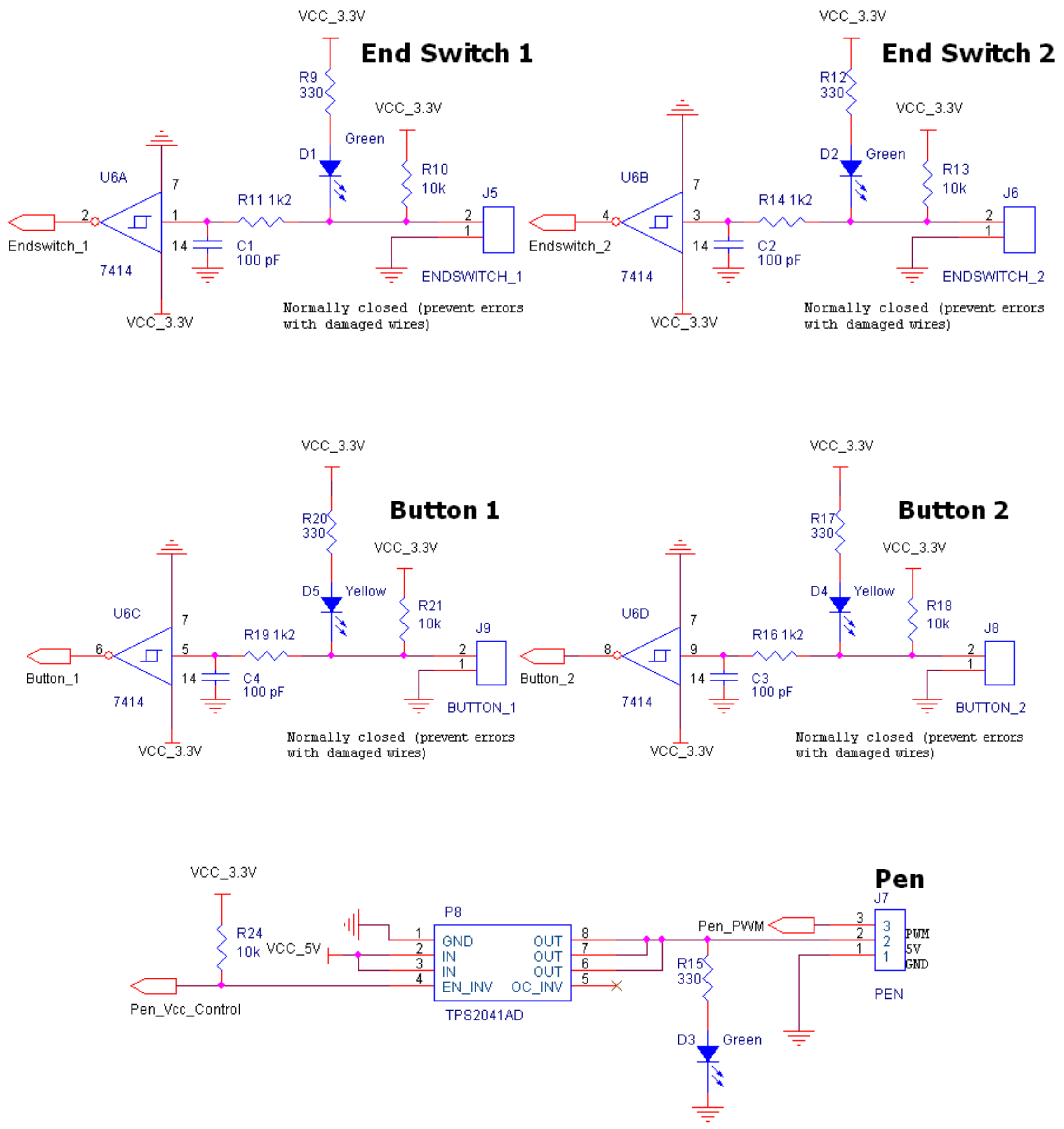
G.4 Connectors pin numbering

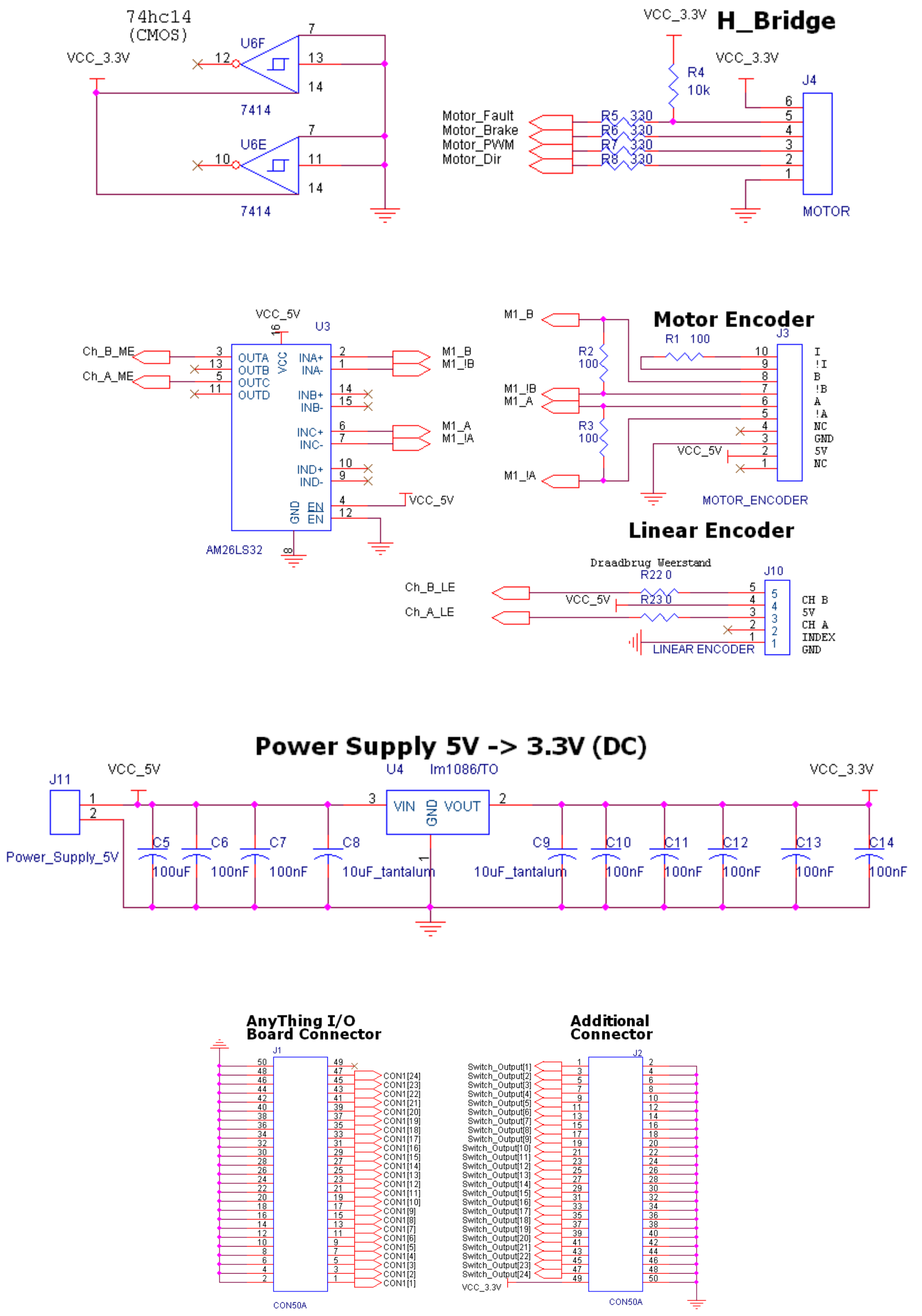
The numbering of the pins can be found in Appendix H, where the electronic circuit is depicted.

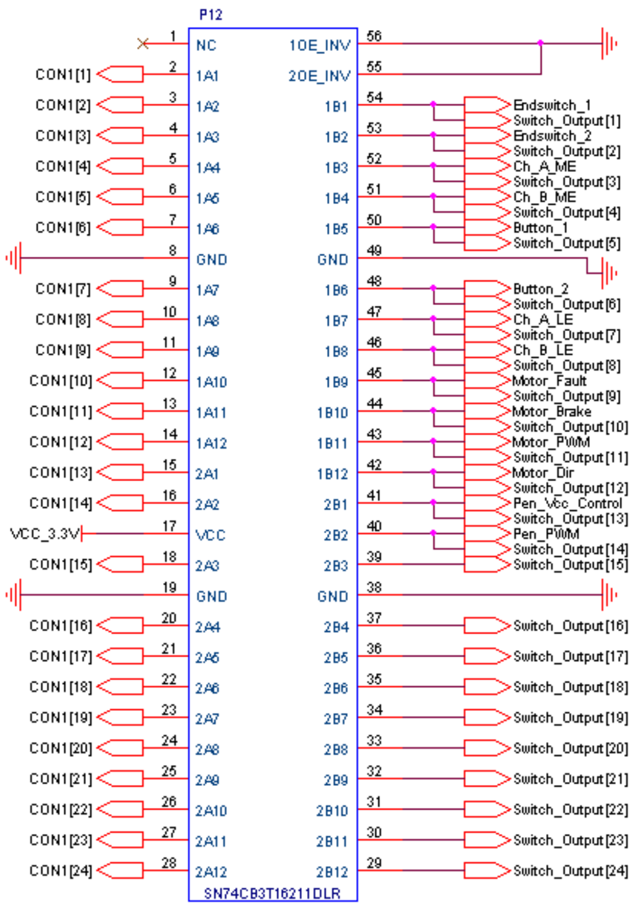
G.5 Errata

The current version of the MCB design contains one error. The 5V DC power supply is not connected with the Vcc_5V supply used at the components. The solution is to connect a engineering wire from the 5V power supply to a Vcc_5V connection.

H Motor Control Board Schematics




74hc14
(CMOS)
VCC_3.3V
U6F
U6E
7414
H_Bridge
R4
10k
Motor_Fault
Motor_Brake
Motor_PWM
Motor_Dir
R5 330
R6 330
R7 330
R8 330
J4
MOTOR
VCC_5V
U3
Ch_B_ME
Ch_A_ME
OUTA
OUTB
OUTC
OUTD
VCC
INA+
INA-
INB+
INB-
INC+
INC-
IND+
IND-
GND
EN
AM26LS32
M1_B
M1_!B
M1_A
M1_!A
Motor Encoder
R1 100
R2 100
R3 100
J3
I
!I
B
!B
A
!A
NC
GND
5V
NC
MOTOR_ENCODER
Linear Encoder
Draadbrug Weerstand
R22 0
R23 0
Ch_B_LE
Ch_A_LE
J10
CH B
5V
CH A
INDEX
GND
LINEAR ENCODER
Power Supply 5V -> 3.3V (DC)
J11
Power_Supply_5V
U4
lm1086/TO
VIN
VOUT
C5 100uF
C6 100nF
C7 100nF
C8 10uF_tantalum
C9 10uF_tantalum
C10 100nF
C11 100nF
C12 100nF
C13 100nF
C14 100nF
AnyThing I/O
Board Connector
J1
CON1[1] … CON1[24]
CON50A
Additional
Connector
J2
Switch_Output[1] … Switch_Output[24]
VCC_3.3V
CON50A


P12
NC
1OE_INV
2OE_INV
1A1
1A2
1A3
1A4
1A5
1A6
GND
1A7
1A8
1A9
1A10
1A11
1A12
2A1
2A2
VCC
2A3
GND
2A4
2A5
2A6
2A7
2A8
2A9
2A10
2A11
2A12
1B1
1B2
1B3
1B4
1B5
GND
1B6
1B7
1B8
1B9
1B10
1B11
1B12
2B1
2B2
2B3
GND
2B4
2B5
2B6
2B7
2B8
2B9
2B10
2B11
2B12
SN74CB3T16211DLR
CON1[1]
CON1[2]
CON1[3]
CON1[4]
CON1[5]
CON1[6]
CON1[7]
CON1[8]
CON1[9]
CON1[10]
CON1[11]
CON1[12]
CON1[13]
CON1[14]
VCC_3.3V
CON1[15]
CON1[16]
CON1[17]
CON1[18]
CON1[19]
CON1[20]
CON1[21]
CON1[22]
CON1[23]
CON1[24]
Endswitch_1
Switch_Output[1]
Endswitch_2
Switch_Output[2]
Ch_A_ME
Switch_Output[3]
Ch_B_ME
Switch_Output[4]
Button_1
Switch_Output[5]
Button_2
Switch_Output[6]
Ch_A_LE
Switch_Output[7]
Ch_B_LE
Switch_Output[8]
Motor_Fault
Switch_Output[9]
Motor_Brake
Switch_Output[10]
Motor_PWM
Switch_Output[11]
Motor_Dir
Switch_Output[12]
Pen_Vcc_Control
Switch_Output[13]
Pen_PWM
Switch_Output[14]
Switch_Output[15]
Switch_Output[16]
Switch_Output[17]
Switch_Output[18]
Switch_Output[19]
Switch_Output[20]
Switch_Output[21]
Switch_Output[22]
Switch_Output[23]
Switch_Output[24]

I Case Study: Failure Analysis at the ViewCorrect Plotter Setup

The described workflow of failure analysis in Chapter 6 is demonstrated in this appendix.

Step 1: Identify Relationship Parts

The ViewCorrect Plotter setup can be divided into seven system levels. Figure I.1 shows the different levels. The lowest level is the mechanical construction. The actuators and sensors are the DC motors, motor encoders, linear encoders, end switches. The printed circuit board processes the signals form the previous level to the Anything I/O board. The Anything I/O board has a FPGA onboard. This board is connected to the PC/104. The plotter software runs on the PC/104. The plotter software is generated and compiled at the development system.

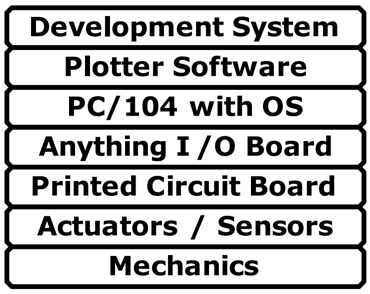


Figure I.1: Different system levels of the ViewCorrect Plotter.

1st level Mechanics	:X-,Y- and Z-Axis
2th level Actuators/Sensors	:Maxon 70W DC Motor, Maxon 150W DC Motor, Servo 5V DC Motor, 2 Motor encoders, 3 Linear encoders, 6 End switches.
3th level Printed Circuit Board	:2 H-Bridges, 3 Motor Control Boards.
4th level Anything I/O Board	:The FPGA is configured with a bit-file designed in the Xilinx software tool. The FPGA configuration includes components which generate PWM signals, read encoders pulses etc. The components can be programmed with registers.
5th level PC/104 with OS	:The PC/104 is an Intel-based computer. The OS is Linux with a Real Time package. The PC/104 communicates with the Anything I/O Board by a driver.
6th level Plotter Software	:The plotter software is generated with 20-Sim and/or gCSP. The software runs on the OS and writes the registers of the FPGA configuration.
7th level Development System	:PC with software tools like Xilinx, 20-Sim, gCSP. The software code is written and/or generated and compiled to executable plotter software or a FPGA configuration.

Figure I.2 shows the relationships between the different levels. The grey blocks are not used at the current setup, but this is possible in future use.

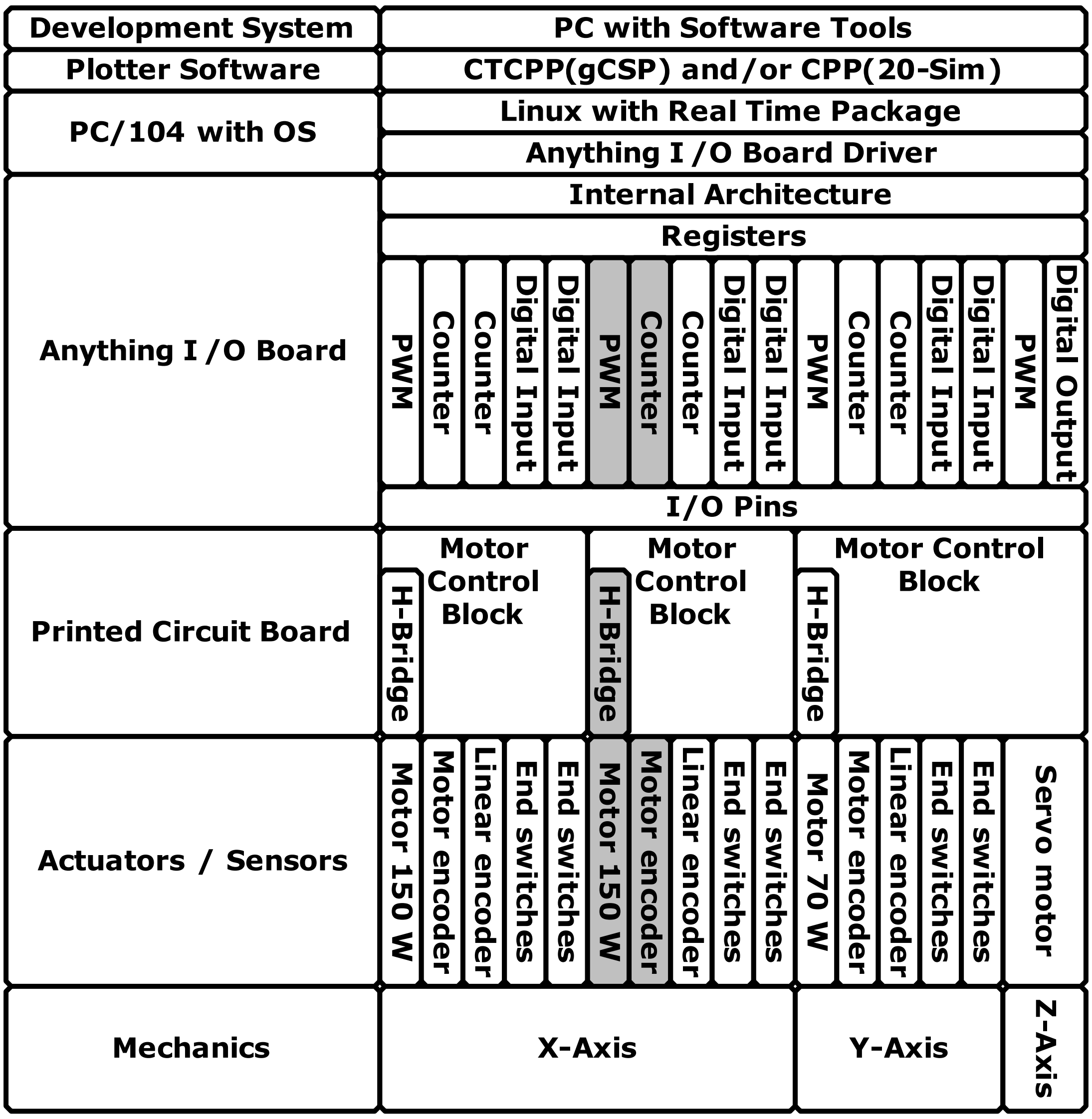


Figure I.2: Relationships between the different system levels of the ViewCorrect Plotter.

Step 2: Identify Failure (Modes)

Each of the individual blocks might fail. Consequently each single component, some components are used several times, has to be analysed which failures are likely to happen.

Mechanics	:Some of the axes can not perform their movements.
Actuators/Sensors	:The motors do not rotate. Encoders and end switches do not detect movement or a touch.
Printed Circuit Board	:H-Bridges and motor control blocks do not process their signals.
Anything I/O Board	:Components do not read input signals or do not write output signals.
PC/104 with OS	:OS, drivers or RT-Package are not correct. OS is not (hard) real-time.
Plotter Software	:Software executable does not work according specifications. Problems with the software timing scheduler.
Development System	:Software tool does not work according specifications. Wrong compiler. Wrong implementation of predefined code of, for example, 20-Sim's real-time toolbox.

Step 3: Determine Causes of Failure

This means analysing the environment, equipment, material or procedures.

Mechanics	:Materials are broken, dirty or bent. Heavy vibration.
Actuators/Sensors	:Materials or equipment are broken. Power supply is broken or utility grid is offline. High operating temperature. Dirty optical transmitter or receiver. Magnetic/electrical radiation.
Printed Circuit Board	:Materials are broken. Power supply is broken or utility grid is offline. Magnetic/electrical radiation. Components are broken. High operating temperature. Loose cables or contacts.
Anything I/O Board	:Component configuration code error or compilation error. Peripheral electronics of the board are broken. Wrong connection caused by a bitfile which is not consistent with the wiring. Power supply is broken or utility grid is offline.
PC/104 with OS	:Wrong distribution of Linux.
Plotter Software	:Code error or compilation error. Wrong implementation of CT-library. Unstable controller software.
Development System	:Wrong installation or software version. Wrong predefined code. Not a correct model of the setup, which causes wrong design decisions or unstable controllers.

Step 4: Determine Consequences of Failures

In this step the effects of failure on the same, higher and overall system level have to be considered. Figure I.2 show the relationships. Failure analysis has to be done from the beginning of a project. If failure analysis occur not until in case of unexpected behaviour or failure the results of step four is outweighed by the work done. In this case, it takes a lot of time to fully analyse all the consequences, but will not give more advantage in solving the failure. Some examples of step four are given.

If for example the motor of the y-axis is broken, the motor and linear encoders detect no movement of the motor and of the y-axis construction. The electronic print boards still processes the signals to and from the Anything I/O board. The plotter software reads the counter register and detects the y-axis is not moving. Consequently the controller will send a PWM signal with a higher duty cycle to reach the set point. Eventually the controller maximize the PWM signal, but the movement is not correct.

If for example the mechanical construction of the y-axis is not able to perform the intended movement, the plotter software reads the counter register and detects the y-axis is not moving. This causes the controller to send a higher duty cycle, which will harm the construction even more.

Step 5: Identify Failure Detection Means

This step aims at addressing ways to detect the identified failures. The approach is to check if the component is according specifications, if not a failure has detected. Existing reports, models or software can be used for testing the specifications.

Mechanics	:Verify the movements and construction of the X-, Y- and Z-Axis by visual or manual inspection.
Actuators/Sensors	:Use a signal analyser, power supply and/or a signal generator to verify the performance characteristics are according the datasheet. Use the mechanical level for testing environment.
Printed Circuit Board	:Use a signal analyser, power supply, actuators/sensors and mechanical level to verify the PCB is according the datasheet.
Anything I/O Board	:Verify the configuration is according the specifications in a Hardware In the Loop Simulation (HILS). If this HILS environment is not available, another quick way is to read and write the I/O ports with a scope analyser and signal generator. With reading and writing the registers the performance characteristics can be verified.
PC/104 with OS	:Verify the distribution and packages are correct. Repair or update the software.
Plotter Software	:Verify the software is according the specifications in a Software In the Loop Simulation (SILS) or co-simulation. Verify the functions used to read and write the registers.
Development System	:Verify the software version is known as a correct one. Repair or update the software. Verification of the model and controller.

Step 6: Detect (Non)-Failure

In this step the actual identification takes place, like described in step 5. Figure I.2 can be used to report the analysed components with marking the block green or red. Step 4 gives the effects of the failure.

J Data Transport Protocols: TCP and UDP

This appendix describes two main transport protocols of the Open Systems Interconnection (OSI) model; TCP and UDP.

J.1 Introduction OSI-model

Ethernet is part of a hierarchal protocol stack. This is separated into different layers. In Figure J.1, a schematic view of the OSI model is given. For all layers a couple of examples are given. Ethernet is part of the data link layer. The IP layer provides a packet based delivery service with no delivery guarantee. On top of this layer a couple of transport protocols exist. In essence two protocols are used: TCP and UDP. The application layer can use UDP or TCP dependent on the requirements.

Application	**HTTP, FTP, POP3**
Presentation	**MPEG, ASCII**
Session	**SDP, NetBios**
Transport	**TCP, UDP, SCTP**
Network	**IPv4, IPv6**
Data Link	**Ethernet, Wi-Fi**
Physical	**Components**

Figure J.1: Schematic view OSI model with some examples.

J.2 Transmission Control Protocol (TCP)

TCP enables two host processes to make a connection and exchange a stream of data with delivery guarantee. Besides the data is delivered is the same sequence as send. The data stream is a sequence of bytes and normally data boundaries are not apparent. Consequently the receiver does not know how the data was send. The data can be fragmented into several pieces. This depends for instance on router algorithms or bandwidth load. Errors will be automatically corrected or retransmitted. It will be notified if an error can not be corrected.

J.2.1 Advantages

- Error correction is done by the OS. Consequently flow control, acknowledgements etc. are done by the kernel and fewer context switches to user space are needed.
- Data is received in same sequence and quantity as send.
- Able to detect congestion of the network and will automatically adjust the transmission speed.

J.2.2 Disadvantages

- It might be inefficient, because it is optimized for certain requirements.
- Large overhead.
- Broadcasting messages is not possible.
- Servers need separate socket for each client.

J.3 User Datagram Protocol (UDP)

UDP provides a connectionless or transaction protocol which enables two host processes to send and receive data without delivery guarantee. The data can be delivered multiple times, in a wrong sequence or it stays even undelivered. The failure rate depends for instance on bandwidth load or router algorithms. UDP provides only a few error recovery services. Due to small packet header UDP has minimal overhead. Data can be sent directly without negotiation or preparation and the data boundaries are preserved. Broadcasting of messages is possible.

J.3.1 Advantages

- Data boundaries are preserved.
- Broadcasting is possible.
- Fast data transmission speed is possible.
- No start up delay, due to connectionless protocol.
- Only implementation of the features the application needs.

J.3.2 Disadvantages

- Unreliable.
- Application has to able to do error correction in case of missing, duplicate or wrong sequence of data.
- No flow control or acknowledgements.

K Manual Programming the Anything I/O Board

This Appendix describes how to program the FPGA configuration on the Anything I/O board manually. This can be necessary for testing or debugging purposes.

The first step is to configure the FPGA with a bit file. The latest bit file customized for the ViewCorrect Plotter setup can be found at:

CeWiki website http://ce226.ewi.utwente.nl/wiki/index.php/WebSVN_repository

Copy this file to the temporarily directory of the PC/104 stack used at the setup. Configuration of the FPGA can be done with the following command. This command is only supported with the second version of Anything I/O board driver.

```
pany /tmp/plotter.bit
rany //Removes the FPGA configuration
```

The second step is to initialize some registers. Linux functions *outw* and *inw* are used to write or read a register. These commands need the hex values of the register and instruction values as arguments. The letter x stands for the specific output or input. The current FPGA configuration supports among other things 6 encoder inputs, 6 PWM outputs, 12 end switches and 6 digital outputs. More information can be found in the regmap.txt file, available at the SVN website.

Global
```
outw 1852 3 //Global Mode Register
```
PWM
```
outw 184x 9 //PWM Control Register
```
Encoder
```
outw 182x 48 //Counter Control Register
```
Digital Out
```
outw 1868 FFFF //Output Register Data
outw 186A FFFF //Output Register Tristate Enable Bits
```
End Switches

Need no initialization

The second step is to write a value to a register for output control or read a register.

PWM
```
outw 187x value //PWM Frequency Control Register
outw 183x value //PWM Output Value Register
```
Encoder
```
inw 180x //Reading Counter Value Register
```
Digital Out
```
inw 1868 //Reading Output Register Data
```
End Switches
```
inw 1862 //Reading End Switch Register
```

Calculating the PWM Output Value

$$\text{DutyCycle}(\%) * 2047 = \text{value} \rightarrow (\text{bin})\text{value} \rightarrow \text{value} << 4 = (\text{hex})\text{register value} \tag{K.1}$$

Example (Duty Cycle 50%)

$$0.5 * 2047 = 1023.5 \rightarrow 1111111111 \rightarrow 11111111110000 = 3\text{FF}0 \tag{K.2}$$

Example values: 7FFO - 100%, 3FF0 - 50%

Calculating the PWM Frequency Control Value

$$\text{Output Frequency} = \left(\frac{\text{value}}{65536}\right) * 16,3\text{kHz} \tag{K.3}$$

Example (Frequency 50 Hz)

$$\text{hex}\left(\frac{50}{16300}\right) * 65536 = \text{C9} \tag{K.4}$$

Bibliography

Astrom, K. J. and B. Wittenmark (1997), *Computer Controlled Systems: Theory and Design*, Prentice Hall.

van den Berg, L. (2006), Design of a Production Cell Setup, MSc Thesis 016CE2005, Control Laboratory, University of Twente.

ten Berge, M. H. (2005), Design Space Exploration for Fieldbus-based Distributed Control Systems, MSc Thesis 029CE2005, Control Laboratory, University of Twente.

Blanchard, B. and W. Fabrycky (2004), Systems Engineering and Analysis 4th edition, Prentice Hall, chapter 6, 12.4, 14.4, ISBN 0131869779.

Buit, E. (2005), PC104 stack mechatronic control platform, MSc Thesis 009CE2005, Control Laboratory, University of Twente.

Cécile, J.-F., L. Schoen, V. Lapointe, A. Abreu and J. Bélanger (2006), A Distributed Real-Time Framework For Dynamic Management Of Heterogeneous Co-simulations, in *Opal-RT Technologies Inc.*, Montréal, Canada.

CERN (2007), HP-GL Viewer, URL `http://service-hpglview.web.cern.ch`.

CLP (2007), Controllab Products B.V., URL `http://www.20sim.com`.

CoWare (2007), CoWare Products, URL `http://www.coware.com/products`.

DDS (2007), OMG Data-Distribution Service for Real-Time Systems, URL `http://www.omg.org/cgi-bin/doc?mars/2003-01-05`.

Deen, B. (2007), CT - I/O Linkdriver with COMEDI, PreDoc Report 009CE2007, Control Laboratory, University of Twente.

Dirne, H. (2005), Demonstrator of advanced controllers, MSc Thesis 013CE2005, Control Laboratory, University of Twente.

EMA (2007), PSpice MATLAB/Simulink Interface, URL `http://www.ema-eda.com/products/orcad/tech.matlabsimulink.aspx`.

FDR2 (2007), Formal Systems (Europe) Ltd, URL `http://www.fsel.com`.

Gheorghe, L., F. Bouchhima, G. Nicolescu and H. Boucheneb (2007), Formal Definitions of Simulation Interfaces in a Continuous/Discrete Co-Simulation Tool, Article, Computer Science Department, Ecole Polytechnique Montréal.

Groothuis, M. A. (2004), Distributed HIL Simulation for Boderc, MSc Thesis 020CE2004, Control Laboratory, University of Twente.

Groothuis, M. A., J. Huang, J. F. Broenink and H. Corporaal (2006), ViewCorrect: Predictable Co-design for distributed embedded mechatronic control systems, in *Progress Mini Symposium I: Embedded System Design*, STW / Progress, p. 1 poster.

Heidenhain (2006), LIDA 4-serie, Heidenhain, URL `http://www.heidenhain.com`.

Hilderink, G., A. Bakkers and J. Broenink (2000), A Distributed Real-Time Java System Based on CSP, in *The third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing ISORC 2000*, IEEE, Newport Beach, CA, pp. 400–407.

Hitec (2007), HS-5125MG Slim Digital Servo, Hitec RCD USA, URL `http://www.hitecrcd.com`.

Hoare, C. (1985), *Communicating Sequential Processes*, Prentice Hall International Series in Computer Science, Prentice Hall.

Jovanovic, D. (2006), PhD thesis, Designing dependable process-oriented software, a CSP approach, Control Laboratory, University of Twente, p. 40.

Jovanovic, D. S., B. Orlic, G. K. Liet and J. F. Broenink (2004), gCSP: A Graphical Tool for Designing CSP systems, in *Communicating Process Architectures*, IOS press, Oxford, UK, pp. 233–251, ISBN 1586034588.

Kuppeveld, T. v. and E. Sprik (2006), Design and Realization of the ViewCorrect Plotter, PreDoc Report 033CE2006, Control Laboratory, University of Twente.

Maljaars, P. (2006), Controllers for the Production Cell Setup, MSc Thesis 039CE2006, Control Laboratory, University of Twente.

Mesanet (2007), 4I65 FPGA based PC104-PLUS Anything I/O card, Mesa Electronics, URL `http://www.mesanet.com`.

Nicolescu, G., H. Boucheneb, L. Gheorghe and F. Bouchhima (2007), Methodology for Efficiënt Design of Continuous/Discrete-Events Co-Simulation Tools, Article, Computer Science Department, Ecole Polytechnique Montréal.

Oce (2007), Océ-Nederland B.V., URL `http://www.oce.com`.

Polis (2007), Hardware-Software Co-Design Group, URL `http://embedded.eecs.berkeley.edu/research/hsc`.

Posthumus, R. (2006), Target Connector, A Redesign of the Hardware Connector, PreDoc Report 028CE2006, Control Laboratory, University of Twente.

Ptolemy (2007), The Ptolemy Project, URL `http://ptolemy.eecs.berkeley.edu`.

Seco (2007), Seco, URL `http://www.seco.it`.

US-Digital (2006), HEDS 9200, US Digital, URL `http://www.usdigital.com`.

VCI (2007), Hardware software interface design, URL `http://www.easics.com`.

Wikipedia (2007a), Kriging, URL `http://en.wikipedia.org/wiki/Kriging`.

Wikipedia (2007b), Shannon's interpolation formula, URL `http://en.wikipedia.org/wiki/Whittaker-Shannon_interpolation_formula`.

Zeigler, Bernard P., H. P. and T. G. Kim (2000), Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, Academic Press, San Diego.