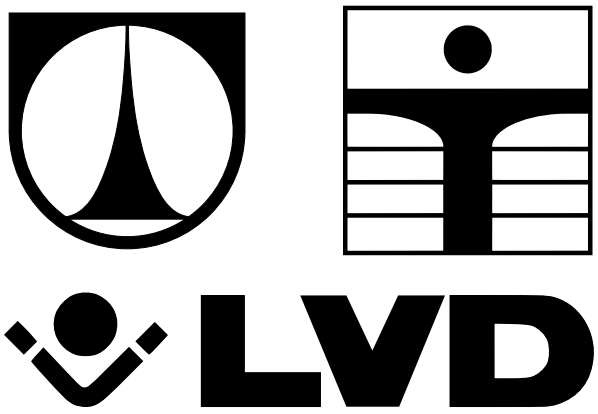
TECHNICAL UNIVERSITY OF LIBEREC

HOCHSCHULE ZITTAU/GÖRLITZ

LVD COMPANY



Application of servo drives on the prototype of the punch press machine

Diploma thesis

Liberec 2014 Bc. Jiří Kuba

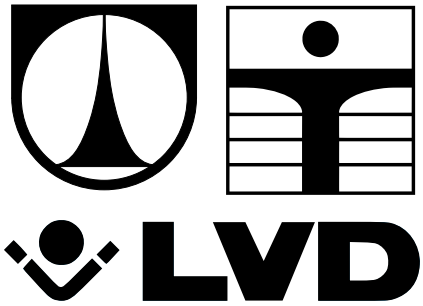
TECHNICAL UNIVERSITY OF LIBEREC

Faculty of Mechatronics, Informatics and Interdisciplinary Studies

HOCHSCHULE ZITTAU/GÖRLITZ

Faculty of Electrical Engineering and Computer Science

LVD COMPANY



Application of servo drives on the prototype of the punch press machine

Diploma thesis

Study programme:	N2612 – Electrical Engineering and Informatics
Study branch:	3906T001 – Mechatronics
Author:	**Bc. Jiří Kuba**
Assessor:	Prof. Dr. Ing. Wolfgang Kästner
Supervisor:	Dr. ir. Wim Serruys

In Liberec 1th October 2014

Declaration

I hereby certify that I have been informed the Act 121/2000, the Copyright Act of the Czech Republic, namely § 60 - Schoolwork, applies to my master thesis in full scope.

I acknowledge that the Technical University of Liberec (TUL) does not infringe my copyrights by using my master thesis for TUL's internal purposes.

I am aware of my obligation to inform TUL on having used or licensed to use my master thesis; in such a case TUL may require compensation of costs spent on creating the work at up to their actual amount.

I have written my master thesis myself using literature listed therein and consulting it with my thesis supervisor and my tutor.

Concurrently I confirm that the printed version of my master thesis is coincident with an electronic version, inserted into the IS STAG.

Date: 1th October 2014

Signature:

Abstract

The master thesis deals with the practical application of servo drives on the prototype of the punch press machine called Dynapunch.

In the first part of the thesis the punching process, Dynapunch and its components are described. Later on the state machine for driving Dynapunch is introduced, coded and tested. It is followed by reprogramming the trajectory generator to the more useful form.

The next part of the thesis focuses on the implementation of the decision algorithm for trajectory adjusting. It requires the programming of an effective solver of motion equations. The next task is the creation of the method for axes motion anticipation. This results in the implementation of the synchronization algorithm for horizontal and vertical axes of the punch press. The algorithm is tested in various conditions. The following part of the thesis describes issues with synchronization of two servo drives connected to the one shaft. The proper solution is proposed and tested.

In the last part of the thesis methods for tuning controllers are described. The manual tuning is used as well as off-line tuning with usage of the transfer function. The transfer function describes the dynamic behaviour of Dynapunch. The transfer function is obtained by system identification. Results of off-line tuning are compared with results provided by manual tuning.

Dynapunch is turned into a machine capable of performing test patterns with a wide set of configuration parameters. The thesis aims for results proved by tests on Dynapunch.

Key words

Punch press, servo drive, real time PC, state machine, cascade control, PID controller, system identification.

Abstrakt

Diplomová práce se zabývá praktickou aplikací servopohonů na prototypu vysekávacího lisu. V první části diplomové práce je popsán proces lisování a lis samotný. Následně je navržen, naprogramován a otestován stavový automat pro řízení lisu. Poté je přepracován generátor trajektorie do lépe použitelné formy.

Další část diplomové práce se zaměřuje na implementaci rozhodovacího algoritmu pro přizpůsobení trajektorie vertikální osy lisu. To vyžaduje naprogramovat efektivní algoritmus pro řešení pohybových rovnic. Dalším úkolem je vytvoření metody pro předpovídání trajektorie os lisu. To vede k implementaci synchronizačního algoritmu pro horizontální a vertikální osu lisu. Algoritmus je testován v různých podmínkách.

V poslední části práce je popsána metoda pro ladění regulátorů servopohonů. Je použita metoda ručního ladění a metoda ladění pomocí přenosové funkce. Přenosová funkce, která popisuje chování lisu, je získána pomocí metody identifikace systému. Výsledky ručního ladění jsou porovnány s výsledky ladění pomocí získané přenosové funkce.

Prototyp lisu je uveden do funkčního stavu, kdy je schopný provádět základní testovací rutiny s širokou volbou parametrů. Výsledky diplomové práce jsou testovány na prototypu lisu.

Klíčová slova

Vysekávací lis, servopohon, operační systém reálného času, stavový automat, kaskádní řízení, PID regulátor, identifikace systému.

Acknowledgment

Foremost, I would like to express my sincere gratitude to my advisor Dr. ir. Wim Serruys for the continuous support during my internship in LVD Company. I wish to express my sincere thanks to Prof. Dr. Ing. Wolfgang Kästner from Hochschule Zittau/Görlitz for providing me all the necessary. Many thanks belongs to ir. Cedric Herreman for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time. I would like to thank my parents for their unconditional support, both financially and emotionally throughout my degree. Last but not least, I would like to give many thanks to my beloved Verunka.

Contents

List of Figures

List of Tables

List of acronyms

AC Altenating Current

API Application Programming Interface

BEMF Back electromotive voltage

BLDC Brushless DC

CF compact flash

CSP Cyclic Synchronous Position mode

DC Direct Current

EtherCAT Ethernet for Control Automation Technology

HMI Human-Machine Interface

IGBT insulated-gate bipolar transistor

MB mega byte, 1 MB = 1024 kb

PDF Portable Document Format

PID proportional-integral-derivative

PWM pulse-width modulation

RMS root mean square

SPM strokes per minute

USB universal serial bus

List of used symbols

Symbol	Description	Unit
a	acceleration	$[\mathrm{ms}^{-2}]$
β	motor viscous damping	$[\frac{\mathrm{Nm \cdot s}}{\mathrm{rad}}]$
C	circumference	[m]
$C(s)$	transfer function of a PI controller	[-]
D	dimameter	[mm]
d	distance	[mm]
e	Euler's number	[-]
F	force	[N]
$H(s)$	transfer function in laplace domain	[-]
I	current	[A]
J	moment of inertia	$[\mathrm{kg \cdot m}^2]$
j	jerk	$[\mathrm{ms}^{-3}]$
k_d	derivative gain	[-]
k_i	integral gain	[-]
k_p	proportional gain	[-]
k_e	BEMF constant	[Vs/rad]
k_t	motor torque constant	[Nm/A]
L	inductance	[H]
N	sample number	[-]
ω, Ω	angular velocity	[rad/s]
ω_0	natural frequency of the system	$[\mathrm{rad \cdot s}^{-1}]$
R_m	tensile strength	[MPa]
$\mathbb{R}$	real numbers	[-]
S	thickness of the sheet	[m]
s	laplace complex argument	[-]
t	time	[s]
T	torque	[Nm]
Θ	angular position	[rad]
U	voltage	[V]
v	velocity	$[\mathrm{ms}^{-1}]$
ζ	relative damping	[-]

1 Introduction

The first punch presses were powered by fly wheels. Later on hydraulics were used. The next step in development is the use of servo drives. Servo drives benefits are cost reduction, easier maintenance and lower energy consumption compared to hydraulic systems. Servo drives are becoming the domain of low tonnage punch presses while hydraulic is suitable for medium and high tonnage presses.

The thesis discuses implementation of servo drive technology on the prototype of a punch press machine. The prototype of the punch press serves as a test platform called Dynapunch. Dynapunch is assembled by three servomotors which are driven by servo drives. Servo drives are commanded by a real time embedded PC. The real time PC controls all actions of Dynapunch.

The thesis consists of four main tasks. The first task is to create an algorithm for performing axis movements. It requires the creation of a complex state machine which is able to perform the testing pattern.

The second task focuses on synchronization of axes to achieve a better strokes per minute ratio. Synchronization is done between two axes. The method for motion anticipation is implemented. The third task aims for synchronization of two servomotors on the same shaft. The Z axis is intended for punching and it is powered by two servomotors which are connected mechanically. However motors do not have interconnected servo drives on the hardware level. This leads to issues with proper motors coordination. The problem is solved by placing part of the control loop outside drives into the real time PC. This allows the control of both servo drives properly.

The last task aims for improvement of controller tuning. At first servo drives are tuned manually. Than the work on the mathematical Simulink model of Dynapunch is started. However, the Simulink model does not reflect the real properties of Dynapunch well, therefore the model is not used for controller tuning. Later on the transfer function describing the dynamic behaviour of Dynapunch is obtained by the system identification technique. Identification is based on the analysis of output signals from the system. The system is excited by the sine sweep.

The transfer function is used for tuning the velocity controller as well as the position

controller. The velocity controller is tuned with application of parametrization of the transfer function. The tuning of the position controller is transferred to the optimization task. The proportional gain of the controller is perpetually changed until the minimum value of the criteria function is found.

It is important to mention that Dynapunch is a prototype in an early phase of development. A lot of minor tasks have to be solved and many bugs have to be found and fixed or reported.

2 Punching

Punching is a method of metal forming. It is the equivalent of shearing. The tool called punch is forced into the material (workpiece). The workpiece is usually a sheet of metal. Depending on the application, a slug or perforated sheet is the product.

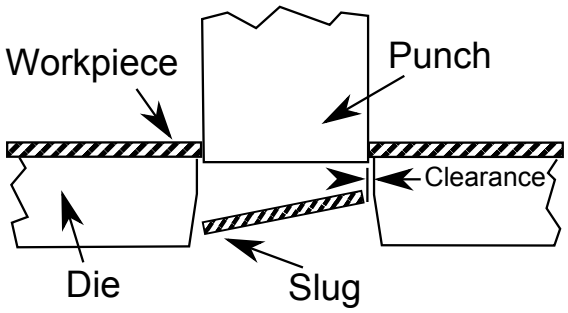


Figure 1: Schematic of the die and the punch

As seen in figure 1, the opposite part of the punch is the die. There is a small space between the punch and the die called clearance. The clearance is necessary to prevent the punch from jamming in the die and also for better quality of hole edge.

According to (Serruys [1]), the force required for perforation of the metal sheet is defined by equation 1 where R_m is tensile strength of the material, S is thickness and C is circumference of the hole.

$$F = R_m \cdot S \cdot C \quad [\mathrm{N}] \tag{1}$$

For example the force required to cut the circular hole with diameter of 3 cm through thick the steel sheet 5 mm thick with $R_m = 450\,\mathrm{N/mm^2}$ is: $F = 450 \cdot 10^6 \cdot 0{,}005 \cdot \pi \cdot 0{,}03 \doteq 211$ kN.

3 Dynapunch description

Dynapunch is the prototype of the punch press. Figure 2 shows the body of Dynapunch. It consists of a bridge frame accompanied by two axes. The Z axis serves for punching. There are two Mavilor BLS-115 servomotors connected to the Z axis. Torque produced by both motors is transferred to the vertical movement by the gearbox and spindle. The theoretical maximum press force is 226 kN. The X axis is for shifting the workpiece. Axis is powered by servomotor Mavilor BLS-73. The momentum provided by the motor is transferred through a belt and spindle to horizontal movement.

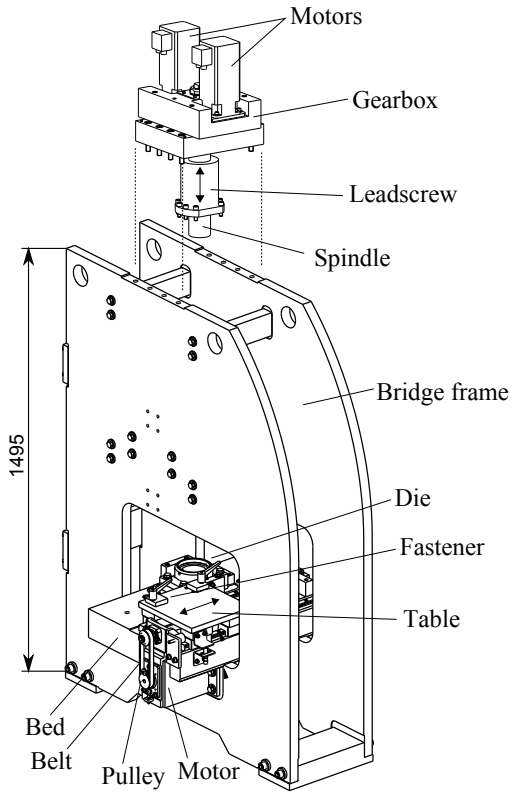


Figure 2: Dynapunch

All motors are driven by servo drives made by IRT SA company. They are custom made for LVD, but very similar to drives in figures 5 and 6. Drives made for LVD have a USB connector for debugging and an EtherCAT interface instead of serial and parallel ports seen in figures. The X axis is driven by servo drive IRT 4009, the Z axis is powered by the pair of stronger IRT 4025. Simple IO logic, such as end switches is implemented by Beckhof digital input terminals [11] with EtherCAT support.

3.0.1 Z axis

The axis is shown on the simplified scheme in figure 3. The axis is assembled by the ram. The ram is connected to the spindle, which is linked with the constant ratio gearbox. The gearbox interconnects two Mavilor servomotors. The gearbox has ratio 21/133 and the spindle ratio is 1/20 mm. It means that ten revolutions of the motor causes $10 \cdot \frac{21}{133} \cdot 20 = 31{,}57$ mm long movement of the axis.

The motion profile of the Z axis is described in chapter 4.2. It can be characterised as perpetual movement from top position called the hover height to the bottom position called the die penetration. The motion has to be finished as fast as possible to keep good strokes per minute (SPM) ratio.

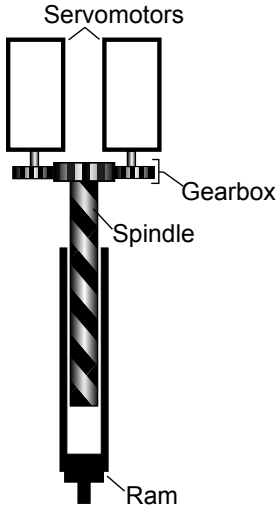


Figure 3: The Z axis scheme

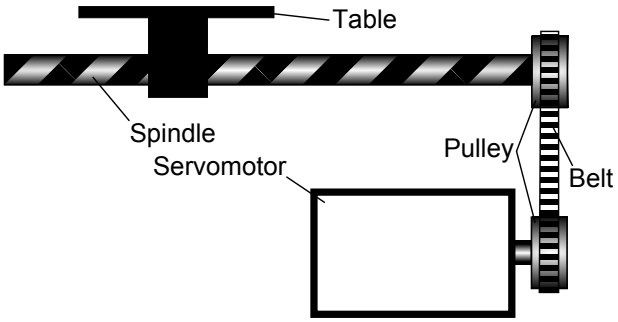


Figure 4: The X axis scheme

3.0.2 X axis

The X axis has a table for shifting the workpiece. The simplified scheme is depicted in figure 4. The axis is driven by servomotor Mavilor BLS-73. The servomotor is connected through the belt to the spindle which moves the carriage with the table. The spindle has ratio 1/20 mm. Pulleys have the same diameter.

The motion profile of the X axis is characterised as a point to point movement with a standstill phase. The movement is done while the punch is above the workpiece. The transition must be as fast as possible to achieve a good strokes per minute (SPM) ratio. The motion profile is described in chapter 4.2.

3.1 IRT servo drives

IRT servo drives are designed for the control of 3 phase brushless servomotors. Servo drives are fully digital, allowing changes to various tuning parameters. They are equipped by safety guards for prevention of motor or drive damage. There is also diagnostic tool which allows to performance of the basic setting on-line through the USB port. Servo drives have IGBT output stage with digital PWM module. It provides low ripple motor current. Drives are controlled through the EtherCAT [3] field bus. The resolver feedback from servomotors is supported.

Figure 5: IRT drive 20 A, source in footnote 1

Figure 6: IRT drive 50 A, source in footnote 2

Servo drives series 4000 AT are used in Dynapunch. There are two driver versions. The smaller version is called 4009 and is used for driving the X axis. It has rated RMS current 9 A. The bigger version is called 4025. It has rated RMS current 25 A. Two IRT 4025 servo drives are used for driving the Z axis. Both versions have AC supply voltage 3×400 V and DC output voltage 3×390 V. Drives are depicted in figure 5. They support several modes of operation.

The Cyclic Synchronous Position mode (CSP) is used in Dynapunch. The CSP mode does not have any trajectory generator, the target position is sent to the drive every 1 ms. The drive is trying to reach target position with the effort defined by setting of control loops and given limits for velocity, acceleration and torque. The control loop scheme is in figure 7. Cascade regulation is used. The position control block consists of the P controller, velocity and torque control blocks are equipped with PI controllers. CSP supports additional offsets such as position offset, velocity and torque offset. It must be noted that *Torque offset* is

[1] `http://www.irtsa.com/spip.php?article3&lang=en`
[2] `http://www.irtsa.com/spip.php?article2&lang=en`

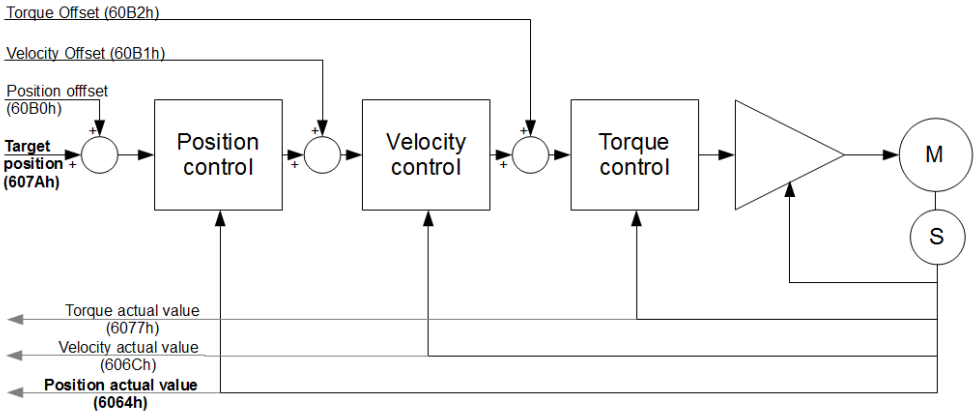


Figure 7: CSP control loop, Source [10]

actually current offset. Name *Torque offset* is used because torque is proportional to the current applied to the motor by equation 7.

Other control parameters are present, but important ones are *Max current* and *Motor rated current*. The parameter *Max current* limits current applied to the motor. It is important to set the proper value according to the motor documentation, otherwise the motor coil insulation could be damaged. Both parameters are also used for calculation of motor overheating factor - I^2T. There is documentation [9] for more details.

Torque and velocity control blocks (figures 8 and 9) are carried out as digital PI controllers.

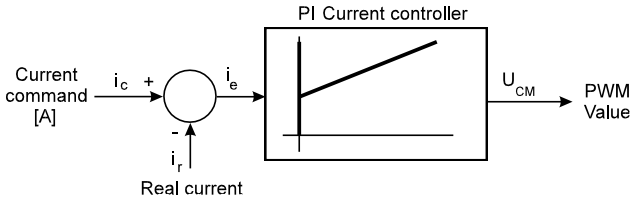


Figure 8: Current control block, source [10]

$$U_{CM}(N) = k_p \cdot i_{e(N)} + k_i \cdot \sum_{i=0}^{N} (i_{e(i)} \cdot \Delta T) \tag{2}$$

PI controller of the torque control block is implemented in the form described by equation 2 and velocity controller is described by equation 3. The variable N is a sample number, ΔT [s] is sampling time, U_{CM} [V] is voltage for PWM voltage source, k_i and k_p are gains, i_e [A] is current error and ω_e [rad/s] is angular speed error.

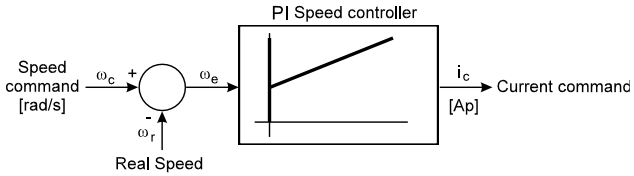


Figure 9: Speed control block, source [10]

$$i_e[N] = k_p \cdot \omega_{e(N)} + k_i \cdot \sum_{i=0}^{N} (\omega_{e(i)} \cdot \Delta T) \tag{3}$$

Position control block (figure 10) consists of proportional controller and feed-forward. It is described by equation 4 where s_c [increments] is the command position and s_e is the position error. Increments are equivalent of the position, they represent output of the resolver built in the motor. However there is no information about the sampling time ΔT.

$$\omega_c(N) = k_p \cdot s_{e(N)} + \frac{s_{c(N)} - s_{c(N-1)}}{\Delta T} \tag{4}$$

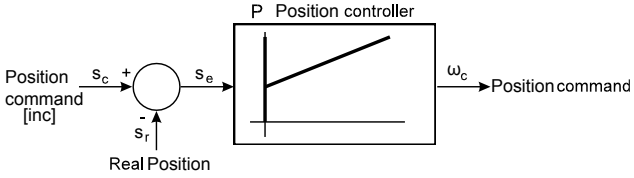


Figure 10: Position control block, source [10]

3.2 Mavilor servomotors

3.2.1 Mavilor BLS-115

Mavilor BLS-115 is the synchronous servomotor with 3 winding phases in the stator. The shaft consists permanent magnets. The motor has brushless construction. Position feedback is produced by the resolver. The motor is characterised by low inertia/torque ratio. It provides very good acceleration characteristic needed for punching with the Z axis. Peak torque is 55,6 Nm, nominal torque is 13,9 Nm and moment of inertia is $0{,}93.10^{-3}\,\mathrm{kg{\cdot}m^2}$. Full specification is in the datasheet [7].

3.2.2 Mavilor BLS-73

Mavilor BLS-73 is same type of motor as Mavilor BLS-115, but smaller. Its peak torque is 10,8 Nm, nominal torque is 2,7 Nm and the moment of inertia is $0{,}74 \cdot 10^{-3}\ \mathrm{kg \cdot m^2}$. Full specification is in the datasheet [8].

3.3 Real time PC

Dynapunch is driven by the real time embedded PC. The real time PC runs on embedded Linux. The Linux consists of kernel 2.4.24 and RTAI patch to make it a real time operating system. There is more information on RTAI documentation [6]. The PC has AMD LX800 processor [5] and the size of memory is 256 MB. The operating system is stored on the CF card.

One of the modules loaded by the system is the program which controls Dynapunch. The embedded PC works with cycle time 1 ms. It means that all output-input actions are performed each millisecond. The embedded PC supports EtherCAT [3]. EtherCAT is a field bus based on Ethernet. The whole infrastructure is depicted in figure 11.

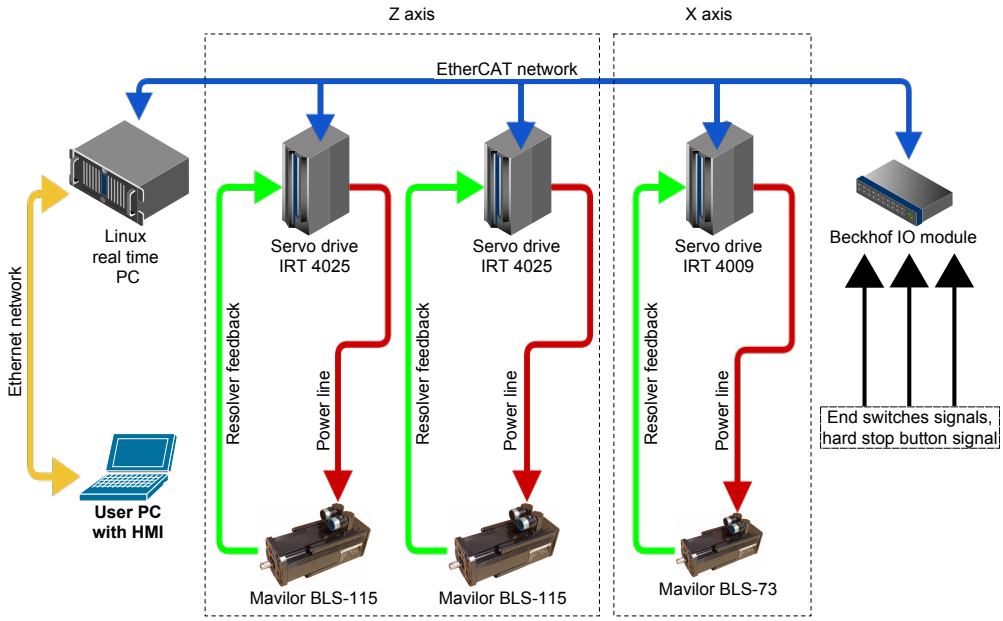


Figure 11: Dynapunch connection scheme

It is possible to command embedded PC via an Ethernet cable. The big advantage of that solution is that Dynapunch is reachable all around the factory if connected to the Intranet

network. A HMI for controlling Dynapunch is a Windows equipped PC with an application called Touch-A developed by LVD. Touch-A is designed for control with a touch screen built in the operator panel.

The module which drives Dynapunch is written in C language. It consist of several parts which are executed sequentially in the specified order. Each part takes care of a specific task. There are tasks such as communication via EtherCAT, control words generator for drives, communication with the Touch-A, state machine implementation and trajectory generation. The source code listed in the thesis is related only to the state machine and trajectory generation. Other parts of the module are not subjects of the thesis.

3.4 DC motor equations

In the thesis are used BLDC motors. A simplified model of the BLDC motor is depicted in figure 12. The model is described by a set of equations. Equations can be divided into an electrical and mechanical part.

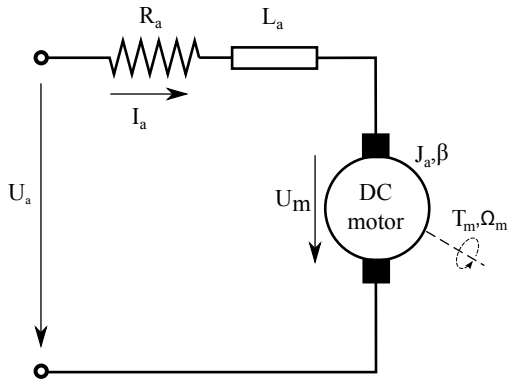


Figure 12: Scheme of a BLDC motor

The electrical part consists of equation

$$U = U_m + R_a I_a + L_a \frac{dI_a}{dt} \tag{5}$$

where U [V] is source voltage, I_a [A] is winding (armature) current, L_a [H] is winding inductance and winding resistance is R_a [Ω]. U_m [V] is back electromotive voltage. U_m is

expressed as

$$U_m = k_e \dot{\Theta} \tag{6}$$

where $\dot{\Theta} = \Omega_m$ [rad/s] is angular velocity of the rotor and k_e [Vs/rad] is BEMF constant. Relation between armature current I_a and torque T_a [Nm] is described by equation 7 where k_t [Nm/A] is motor torque constant.

$$T_a = k_t I_a \tag{7}$$

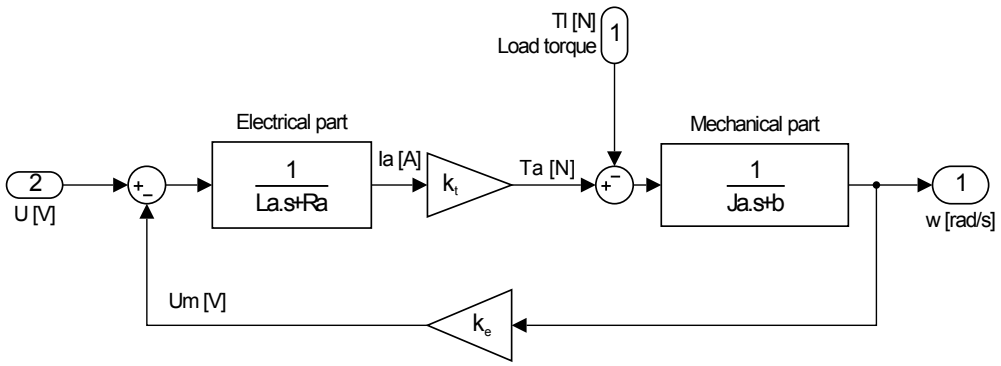


Figure 13: Model of DC motor

Equation 5 can be combined with equation 6 into the form

$$U = k_e \dot{\Theta} + R_a I_a + L_a \frac{dI_a}{dt} \tag{8}$$

Mechanical part is described by equation 9:

$$0 = k_t I_a - J_a \ddot{\Theta} - \beta \dot{\Theta} - T_l \tag{9}$$

where T_l [Nm] is output torque of the motor and J_a [kg $\cdot$ m^2] is moment of inertia. Coefficient β [$\frac{\text{Nm}\cdot\text{s}}{\text{rad}}$] is motor viscous damping. Damping is hard to measure and can be nonlinear, but for simplification it is stated as constant. Taking the Laplace transform of equation 8 and 9 gives:

$$U(s) = k_e s\Theta(s) + R_a I_a(s) + L_a s I_a(s) \tag{10}$$

$$0 = k_t I_a(s) - J_a s^2 \Theta(s) - \beta s \Theta(s) - T_l(s) \tag{11}$$

Equations 10 and 11 are rewritten to get $I_a(s)$ and $\Omega_m(s)$:

$$I_a(s) = \frac{-k_e s\Theta(s) + U(s)}{sL_a + R_a} \tag{12}$$

$$\Omega_m(s) = s\Theta(s) = \frac{k_t I_a(s) - T_l(s)}{sJ_a + \beta} \tag{13}$$

The Simulink model of the DC motor in figure 13 is derived from equations 12 and 13. The model is considered as an approximation of servomotors used in Dynapunch. Transfer functions of BLDC motor scheme are

$$\frac{\Omega_m(s)}{U(s)} = \frac{\frac{1}{L_a s+R_a} k_t \frac{1}{J_a s+b}}{1 + \frac{1}{L_a s+R_a} k_t \frac{1}{J_a s+b} k_e} = \cdots = \frac{k_t}{L_a J_a s^2 + (L_a b + J_a R_a)s + R_a b + k_t k_e} \tag{14}$$

$$\frac{\Omega_m(s)}{T_l(s)} = \frac{-\frac{1}{J_a s+b}}{1 + \frac{1}{L_a s+R_a} k_t \frac{1}{J_a s+b} k_e} = \cdots = \frac{-L_a s + R_a}{L_a J_a s^2 + (L_a b + J_a R_a)s + R_a b + k_t k_e} \tag{15}$$

Transfer functions 14 and 15 can be combined to have Ω_m response on U_m and T_l:

$$\Omega_{m_{sum}}(s) = \frac{\Omega_m(s)}{U(s)} U(s) + \frac{\Omega_m(s)}{T_l(s)} T_l(s) \tag{16}$$

4 Punch cycle

Punch cycle consists of motions of both axes. Motions are described by set of equations combined to motion profiles. Following chapter describes motion profile of the Z axis.

4.1 Motion equations

The motion profile of the Z axis is depicted in figure 14. It consist of 6 zones t1 - t6. The motion profile is defined by jerk j [ms^{-3}], maximum acceleration a_{max} [ms^{-2}] and maximum speed v_{max} [ms^{-1}].

Zone t1	Zone t2	Zone t3	Zone t4	Zone t5	Zone t6
$j_1 = -j$	$j_2 = 0$	$j_3 = j$	$j_4 = j$	$j_5 = 0$	$j_6 = -j$
$t_1 = \frac{a_{max}}{j}$	$t_2 = \frac{v_{max} - 2v_1}{a_{max}}$	$t_3 = \frac{a_{max}}{j}$	$t_4 = t_3$	$t_5 = \frac{a_{max}}{j}$	$t_6 = \frac{v_5}{a_{max}}$

Table 1: Jerk and t_n values for zones t1 - t6

Time t_n [s] spent in n^{th} zone and respective jerk value j_n is listed in the table 4.1. Actual values of acceleration a_n, speed s_n [ms^{-1}] and position s_n [m] for n^{th} zone can be calculated by equations 18, 19 and 20 with initial conditions 17.

$$n \in \{1, 2, 3, 4, 5, 6\} \tag{17}$$

$$t \in (t_{n-1}, t_n) \quad [s]$$

$$t_0 = 0, a_0 = 0, v_0 = 0$$

$$a_n(t) = a_{n-1}(t_{n-1}) + \int_{t_{n-1}}^{t} j_n \, \mathrm{d}t \tag{18}$$

$$v_n(t) = v_{n-1}(t_{n-1}) + \int_{t_{n-1}}^{t} a_n \, \mathrm{d}t \tag{19}$$

$$s_n(t) = s_{n-1}(t_{n-1}) + \int_{t_{n-1}}^{t} v_n \, \mathrm{d}t \tag{20}$$

For example equations for the zone t1 are

$$a_1(t) = \int_{t_0}^{t} j \, \mathrm{d}t = -jt \tag{21}$$

$$v_1(t) = \int_{t_0}^{t} a_1(t)\,\mathrm{d}t = -\frac{jt^2}{2} \tag{22}$$

$$s_1(t) = \int_{t_0}^{t} v_1(t)\,\mathrm{d}t = -\frac{jt^3}{6} \tag{23}$$

Equations for the zone t2 are

$$a_2(t) = a_1(t_1) + \int_{t_1}^{t} j_2\,\mathrm{d}t = -jt_1 \tag{24}$$

$$v_2(t) = v_1(t_1) + \int_{t_1}^{t} a_2(t)\,\mathrm{d}t = -\frac{j{t_1}^2}{2} - jt_1t + jt_1^2 = \frac{j{t_1}^2}{2} - jt_1t \tag{25}$$

$$s_2(t) = s_1(t_1) + \int_{t_1}^{t} v_2(t)\,\mathrm{d}t = -\frac{j{t_1}^3}{6} + \frac{j{t_1}^2}{2}t - \frac{jt_1}{2}t^2 - \frac{j{t_1}^3}{2} + \frac{j{t_1}^3}{2} = -\frac{j{t_1}^3}{6} + \frac{j{t_1}^2}{2}t - \frac{jt_1}{2}t^2 \tag{26}$$

Motion equations can be characterised as polynomial functions up to the 3$^{\mathrm{rd}}$ order.

The following lines describe properties of the motion in the figure 14. When the motion starts, acceleration is limited by maximum acceleration value and built in respect to jerk in the zone t1. The ram goes downwards. Zone t2 has constant acceleration while speed is rising to maximum allowed speed. Zone t3 serves for decreasing acceleration to zero with respect of jerk value. Deceleration is built in the zone t4 with respect of the jerk value. Zone t5 has constant deceleration, speed is decreasing. Deceleration is decreasing to zero in the last zone t6. The ram is in the lowest point of trajectory. The ram is returned to the top position by performing movements through all zones in reverse order.

4.2 Trajectory generator

The trajectory generator generates a trajectory of the Z axis according to motion equations described in the chapter 4.1.

Trajectory generation is divided into two phases. There is an initial phase before the movement starts. Input variables are jerk, acceleration and maximum speed. Generated output is time, velocity and position at the end of each zone.

The second phase starts with the movement. Parameters generated in the initial phase are used to cyclically generate points of trajectory. Which motion equation will be used is determined by pre-generated times for the end of each zone.

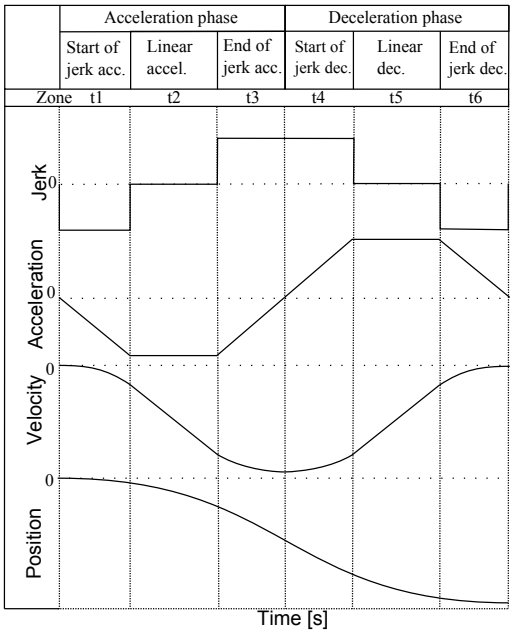


Figure 14: Motion profile of the Z axis - the first half

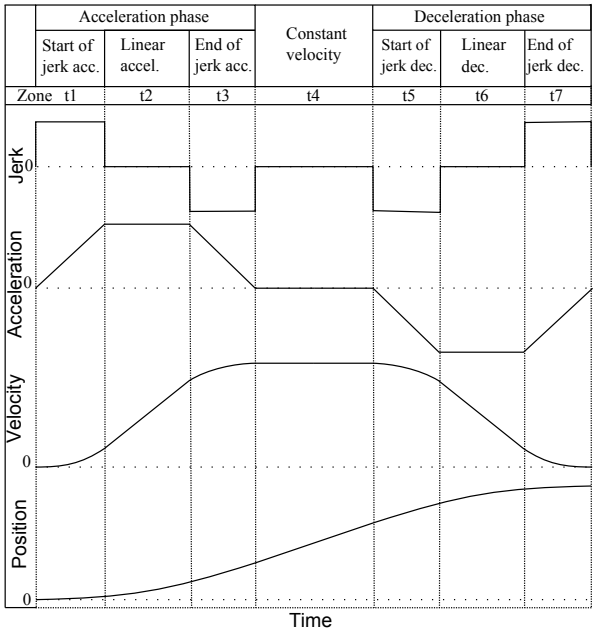


Figure 15: Motion profile of the X axis

Equations for pre-generating during the initial phase are listed in the shortened code listing 1. Code listing contains initial function `init_punch_cycle_movement` where time zone numbers correspond with zones t1 - t6 in figure 14.

Listing 1: Initialization of the trajectory generator

```
1  void init_punch_cycle_movement(ttv_tgen_channel_data* p)
2  {
3      //declare & initialise variables
4      debug_message(DEBUG_TAG,"init_punch_cycle_movement");
5
6      // initial values
7      float j = p->init_data.sfv_jerk/1000.0;
8      float a = p->init_data.sfv_acc/1000.0;
9      float v = p->init_data.sfv_speed/1000.0;
10
11         //time zone 1 - build up acc, limit jerk
12         float t1 = a/j;
13         //float a1 = a;
14         float v1 = j*t1*t1/2.0;
15         float s1 = j*t1*t1*t1/6.0;
16
17         //time zone 2 - max and constant acceleration
18         float t2 = t1 + (v - 2*v1)/a;
19         //float a2 = a;
20         float v2 = v1 + a*(t2 - t1);
21         float s2 = s1 + v1*(t2 - t1) + a*(t2 - t1)*(t2 - t1)/2.0;
22
23         //time zone 3 - acceleration decreases as speed gets higher
24         float t3 = t2 + a/j;
25         //float a3 = 0;
26         float v3 = v2 + a*(t3 - t2) - j*(t3 - t2)*(t3 - t2)/2.0;
27         float s3 = s2 + v2*(t3 - t2) + a*(t3 - t2)*(t3 - t2)/2.0 - j*(t3 - t2)*(t3 - t2)*(t3 - t2)/6.0;
28
29         //time zone 4 - constant speed
```

```
30        float t4 = t3 + 0.0;
31        //float a4 = 0;
32        float v4 = v;
33        float s4 = s3 + (t4 - t3)*v;
34
35        //time zone 5 - build up dec, limit jerk
36        float t5 = t4 + a/j;
37        //float a5 = -j*(t5 - t4);
38        float v5 = v4 - j*(t5 - t4)*(t5 - t4)/2.0;
39        float s5 = s4 + v4*(t5 - t4) - j*(t5 - t4)*(t5 - t4)*(t5 - t4)/6.0;
40
41        //time zone 6 - max and constant deceleration
42        float t6 = t5 + v5/a;
43        //float a6 = -a;
44        float v6 = v5 - a*(t6 - t5);
45        float s6 = s5 + v5*(t6 - t5) - a*(t6 - t5)*(t6 - t5)/2.0;
46        ..... \\shortened
```

Pre-generated values for each time zone are forwarded to function `set_punch_cycle_point_generation` in the code listing 2. The function is called every 1 ms to get a new point of the trajectory. Time t is compared with the end time of zones t1 - t6 to choose which zone will be used. Actual position, speed and acceleration is generated.

Listing 2: Original version of the point generator

```
1  void set_punch_cycle_point_generation(ttv_tgen_channel_data* p,float pfv_time,float* pfp_pos,float* pfp_spd,
       float* pfp_acc)
2  {
3         ..... \\shortened
4         float t = pfv_time;
5         if (pfv_time > t6) \\ if time is behind zone t6, zones are executed backwards
6                t = (2*t6 - t);
7
8     float j = p->init_data.sfv_jerk/1000.0;
9     float a = p->init_data.sfv_acc/1000.0;
10    float v = p->init_data.sfv_speed/1000.0;
11
12        // initialization
13        float jn = 0.0;
14        float an = 0.0;
15        float vn = 0.0;
16        float sn = 0.0;
17
18        if (t <= 0.0) // no motion zone
19        {
20               jn = 0.0;
21               an = 0.0;
22               vn = 0.0;
23               sn = 0.0;
24        }
25        else if (t <= t1) // zones are chosen according to t1 - t6
26        {
27               //time zone 1
28               jn = j;
29               an = j*t;
30               vn = j*t*t/2.0;
31               sn = j*t*t*t/6.0;
32        }
33        else if (t <= t2)
34        {
35               //time zone 2
36               jn = 0.0;
37               an = a;
38               vn = v1 + a*(t - t1);
```

```
39              sn = s1 + v1*(t - t1) + a*(t - t1)*(t - t1)/2.0;
40      }
41      else if (t <= t3)
42      {
43              //time zone 3
44              jn = -j;
45              an = a - j*(t - t2);
46              vn = v2 + a*(t - t2) - j*(t - t2)*(t - t2)/2.0;
47              sn = s2 + v2*(t - t2) + a*(t - t2)*(t - t2)/2.0 - j*(t - t2)*(t - t2)*(t - t2)/6.0;
48      }
49      else if (t <= t4)
50      {
51              //time zone 4 - constant speed
52              jn = 0;
53              an = 0;
54              vn = v;
55              sn = s3 + (t - t3)*v;
56      }
57      else if (t <= t5)
58      {
59              //time zone 5
60              jn = -j;
61              an = -j*(t - t4);
62              vn = v4 - j*(t - t4)*(t - t4)/2.0;
63              sn = s4 + v4*(t - t4) - j*(t - t4)*(t - t4)*(t - t4)/6.0;
64      }
65      else if (t <= t6)
66      {
67              //time zone 6
68              jn = 0;
69              an = -a;
70              vn = v5 - a*(t - t5);
71              sn = s5 + v5*(t - t5) - a*(t - t5)*(t - t5)/2.0;
72      }
73      else
74      {
75              jn = 0.0;
76              an = 0.0;
77              vn = 0.0;
78              sn = 0.0;
79      }
80
81      if (pfv_time > t6)
82      {
83              jn *= -1;
84              vn *= -1;
85      }
86
87      *pfp_acc = -an*1000.0;
88      *pfp_spd = -vn*1000.0;
89      *pfp_pos = p->init_data.sfv_s_start - 1000.0*sn;
90  }
```

The big disadvantage of this approach is that it is not possible to set distance between the top and the bottom point of the trajectory. It is necessary to balance jerk, acceleration and speed to get the desired distance. Therefore the function `set_punch_cycle_point_generation` in the code listing 2 is reprogrammed. The variable `distance` is added to specify desired distance. However if the maximum speed is set too low, it is not possible to reach the desired distance. Therefore the constant speed phase is added together with the decision algorithm to keep trajectory as fast as possible. A shortened code of the new function is in the code

listing 3. The new function generates trajectory with the following properties. If the desired distance is longer than distance reached with motion profile without constant speed phase, the constant speed phase is added (lines 61 - 84). If the desired distance is shorter than distance reached with predefined jerk, acceleration and max. speed values, the constant acceleration phase is shortened so as not to overcome the desired distance (lines 86 - 138). If pre-generated speed exceeds max. speed, it is limited and the constant speed phase is enlarged (lines 18 - 25). The new motion profile is depicted in figure 16.

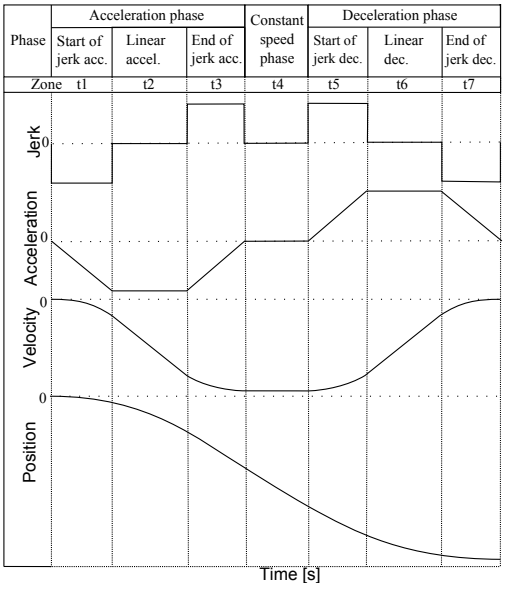


Figure 16: Motion profile of Z axis with constant velocity zone - the first half

Listing 3: Point generator with specified distance and max. speed limitation

```
1  void init_punch_cycle_movement_with_const_speed(ttv_tgen_channel_data* p)
2  {
3      //declare & initialise variables
4      debug_message(DEBUG_TAG,"init_punch_cycle_movement_with_const_speed");
5
6      float j = p->init_data.sfv_jerk/1000.0;
7      float a = p->init_data.sfv_acc/1000.0;
8      float v = p->init_data.sfv_speed/1000.0;
9      float distance = p->init_data.sfv_a_a/1000.0;
10
11     //time zone 1 - build up acc, limit jerk
12         float t1 = a/j;
13         //float a1 = a;
14         float v1 = j*t1*t1/2.0;
15         float s1 = j*t1*t1*t1/6.0;
16
17         // max. speed will be reached during building up acc
18         if(v1>=v/2)
19     {
20         v1 = v/2;
21         t1 = sqrtf(2 * v1 / j);
```

```
22          a = j * t1;
23          s1 = j*t1*t1*t1/6.0;
24          // printf("time of building up acc limited\n");
25      }
26
27          // time zone 2 - max and constant acceleration
28          float t2 = t1 + (v - 2*v1)/a;
29          // float a2 = a;
30          float v2 = v1 + a*(t2 - t1);
31          float s2 = s1 + v1*(t2 - t1) + a*(t2 - t1)*(t2 - t1)/2.0;
32
33          // time zone 3 - acceleration decreases as speed gets higher
34          float t3 = t2 + a/j;
35      if(v1>=v/2) // max. speed will be reached during building up acc
36      {
37         t3 = t2 + t1;
38      }
39          // float a3 = 0;
40          float v3 = v2 + a*(t3 - t2) - j*(t3 - t2)*(t3 - t2)/2.0;
41          float s3 = s2 + v2*(t3 - t2) + a*(t3 - t2)*(t3 - t2)/2.0 - j*(t3 - t2)*(t3 - t2)*(t3 - t2)/6.0;
42
43          // time zone 4 - constant speed
44          float t4 = t3 + 0.0;
45          // float a4 = 0;
46          float v4 = v;
47          float s4 = s3 + (t4 - t3)*v;
48
49          // time zone 5 - build up dec, limit jerk
50          float t5 = t4 + a/j;
51          // float a5 = -j*(t5 - t4);
52          float v5 = v4 - j*(t5 - t4)*(t5 - t4)/2.0;
53          float s5 = s4 + v4*(t5 - t4) - j*(t5 - t4)*(t5 - t4)*(t5 - t4)/6.0;
54
55          //time zone 6 - max and constant deceleration
56          float t6 = t5 + v5/a;
57          //float a6 = -a;
58          float v6 = v5 - a*(t6 - t5);
59          float s6 = s5 + v5*(t6 - t5) - a*(t6 - t5)*(t6 - t5)/2.0;
60
61      if(s6<distance) // constant speed zone is added
62      {
63          debug_message(DEBUG_TAG,"Build constant speed zone");
64
65          float distance_missing = distance - s6;
66
67          //time zone 4 - constant speed
68          t4 = t3 + distance_missing / v4;;
69          s4 = s3 + (t4 - t3)*v;
70
71          //time zone 5 - build up dec, limit jerk
72          t5 = t4 + a/j;
73          //float a5 = -j*(t5 - t4);
74          v5 = v4 - j*(t5 - t4)*(t5 - t4)/2.0;
75          s5 = s4 + v4*(t5 - t4) - j*(t5 - t4)*(t5 - t4)*(t5 - t4)/6.0;
76
77          //time zone 6 - max and constant deceleration
78          t6 = t5 + v5/a;
79          //float a6 = -a;
80          v6 = v5 - a*(t6 - t5);
81          s6 = s5 + v5*(t6 - t5) - a*(t6 - t5)*(t6 - t5)/2.0;
82      }
83
84      if(s6 > distance) // we are beyond desired distance, acceleration zone is shortened
85      {
86          debug_message(DEBUG_TAG,"Shortening const acc phase");
87
88          //float real_s2 = s2 - s1;
89          float ss3 = s3 - s2;
90          // desired s2
91          float desired_s2 = (distance - s1 - ss3)/2;
```

```
92
93
94        //time zone 2 - max and constant acceleration
95          //t2 = t1 + (v - 2*v1)/a;
96        float c[5]={0};
97        c[0] = - desired_s2;
98        c[1] = v1;
99        c[2] = 0.5 * a;
100       float tt2 = tgen_newton(c, 2, 1, 0, 9999); //usage of the Newton method to get t2
101
102       t2 = t1 + tt2;
103       v2 = v1 + a*(t2 - t1);
104       s2 = s1 + v1*(t2 - t1) + a*(t2 - t1)*(t2 - t1)/2.0;
105
106         //time zone 3 - acceleration decreases as speed gets higher
107         t3 = t2 + a/j;
108       if(v1>=v/2) // max. speed will be reached during building up acc
109       {
110            t3 = t2 + t1;
111       }
112         // a3 = 0;
113         v3 = v2 + a*(t3 - t2) - j*(t3 - t2)*(t3 - t2)/2.0;
114         v = v3;
115         s3 = s2 + v2*(t3 - t2) + a*(t3 - t2)*(t3 - t2)/2.0 - j*(t3 - t2)*(t3 - t2)*(t3 - t2)/6.0;
116
117         //time zone 4 - constant speed
118         t4 = t3 + 0.0;
119         //float a4 = 0;
120         v4 = v3;
121         s4 = s3 + (t4 - t3)*v;
122
123         //time zone 5 - build up dec, limit jerk
124         t5 = t4 + a/j;
125         //float a5 = -j*(t5 - t4);
126         v5 = v4 - j*(t5 - t4)*(t5 - t4)/2.0;
127         s5 = s4 + v4*(t5 - t4) - j*(t5 - t4)*(t5 - t4)*(t5 - t4)/6.0;
128
129         //time zone 6 - max and constant deceleration
130         t6 = v5/a + t5;
131         //float a6 = -a;
132         v6 = v5 - a*(t6 - t5);
133         s6 = s5 + v5*(t6 - t5) - a*(t6 - t5)*(t6 - t5)/2.0;
134   }
135
136 ..... \\ shortened
```

4.3 Summary

The code for trajectory generation is rewritten in a more useful form. It is no longer necessary to recompute jerk, acceleration and speed values to get the desired stroke length. If the maximum speed or acceleration is limited or the distance from hover height to the bottom is too long, constant speed phase is automatically inserted to the motion profile to achieve the desired stroke length.

5 State machine

The following chapter refers to the development of the state machine for driving Dynapunch. The state machine started as a simple machine which was performing up and down movements with one axis in an endless loop. The complex state machine is created. It is able to perform the test pattern listed in figure 20 with various settings. A state machine is built above API developed by the LVD Company. It is a single thread application running in an endless loop as within a Linux system module.

The state machine is divided into three blocks. The initial block serves for axes referencing with limit switches and puts axes to defined positions before the punching sequence starts. The punching block performs the punch pattern and the termination block guides Dynapunch to the initial state after the punch pattern is finished.

The source code of the state machine can be seen in the appendix D.1, because it is too long to place it into the thesis. There is a simplified diagram of it in figure 19. It is recommended to read chapter 5.1 first for good understanding.

5.1 Example of the state machine

A simple state machine is described for understanding the complex state machine in the appendix D.1.

Implementation of the state machine has the following rules. Transitions between states are allowed if the condition in the transition block is fulfilled. If the condition is not fulfilled, transition is not made and the condition is tested periodically. Every action of Dynapunch is triggered in the transition block after the condition is fulfilled. It is possible to connect more than one transition block to the one state block. State and transition are depicted in figure 17.

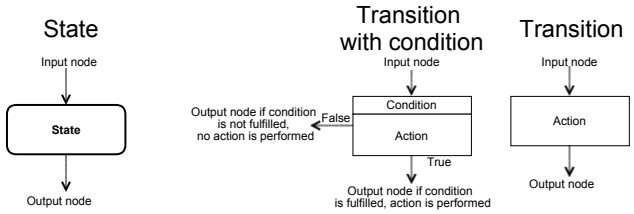


Figure 17: Example of state and transition symbols

Lets assume a state machine in figure 18. It waits in the state 1 until condition 1 is fulfilled. Than it waits in the state 2 until condition 2 is true, then cycles between states 2 and 3 if conditions 2 and 3 are fulfilled. If any condition is not true, the state machine returns to the state 1. Transitions 1,2 and 3 are executed during transitions between states. There can be transition block without a condition, its action is always performed. If there is a node without transition block, transition is done, but no action performed. The source code of the state machine is in the listing 4.

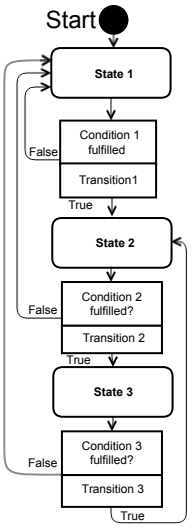


Figure 18: Example of the state machine

Listing 4: Example of the state machine from figure 18

```
1  #include "../../stm/stm_mod.h"
2
3  // state functions prototypes
4  void SMachine_s_1();
5  void SMachine_s_2();
6  void SMachine_s_3();
7
8  // condition function prototypes
9  void SMachine_c_1();
10 void SMachine_c_2();
11 void SMachine_c_3();
12
13 // transition function prototypes
14 void SMachine_t_1();
15 void SMachine_t_2();
16 void SMachine_t_3();
17 void SMachine_t_4();
18
19 //initialization of the state machine
20 int pattern_stm_initialise()
21 {
22     int liv_stm_idx = stm_register_state_machine(); // id of the state machine
23     if (liv_stm_idx == -1)
```

```
24      {
25        debug_message(DEBUG_TAG,"stm_register_state_machine failed");
26        return -1;
27      }
28
29      // register states
30      int liv_state_1 = stm_register_state(liv_stm_idx,SMachine_s_1);
31      int liv_state_2 = stm_register_state(liv_stm_idx,SMachine_s_2);
32      int liv_state_3 = stm_register_state(liv_stm_idx,SMachine_s_3);
33
34      // test states registration
35      if ((liv_state_1 == -1) ||
36          (liv_state_2 == -1) ||
37          (liv_state_3 == -1))
38      {
39        debug_message(DEBUG_TAG,"stm_register_state failed");
40        return -1;
41      }
42
43      // register conditions
44      int liv_cond_1 = stm_register_condition(liv_stm_idx,SMachine_c_1);
45      int liv_cond_2 = stm_register_condition(liv_stm_idx,SMachine_c_2);
46      int liv_cond_3 = stm_register_condition(liv_stm_idx,SMachine_c_3);
47      int liv_cond_4 = stm_register_condition_combination(liv_stm_idx,STM_OPER_NOT,liv_cond_2,0);
48      int liv_cond_5 = stm_register_condition_combination(liv_stm_idx,STM_OPER_NOT,liv_cond_3,0);
49
50      // test conditions registration
51      if ((liv_cond_1 == -1) ||
52          (liv_cond_2 == -1) ||
53          (liv_cond_3 == -1) ||
54          (liv_cond_4 == -1) ||
55          (liv_cond_5 == -1))
56      {
57        debug_message(DEBUG_TAG,"stm_register_condition failed");
58        return -1;
59      }
60
61      // register transitions
62      int liv_trans_1 = stm_register_transition(liv_stm_idx,liv_state_1,liv_state_2,liv_cond_1,SMachine_t_1);
63      int liv_trans_2 = stm_register_transition(liv_stm_idx,liv_state_2,liv_state_3,liv_cond_2,SMachine_t_2);
64      int liv_trans_3 = stm_register_transition(liv_stm_idx,liv_state_3,liv_state_2,liv_cond_3,SMachine_t_3);
65      int liv_trans_4 = stm_register_transition(liv_stm_idx,liv_state_2,liv_state_1,liv_cond_3,SMachine_t_4);
66      int liv_trans_5 = stm_register_transition(liv_stm_idx,liv_state_3,liv_state_1,liv_cond_3,SMachine_t_4);
67
68      // test transitions registration
69      if ((liv_trans_1 == -1) ||
70          (liv_trans_2 == -1) ||
71          (liv_trans_3 == -1) ||
72          (liv_trans_4 == -1) ||
73          (liv_trans_5 == -1))
74      {
75        debug_message(DEBUG_TAG,"stm_register_transition failed");
76        return -1;
77      }
78
79      return 1;
80  }
81
82  //state functions
83  void SMachine_s_1()
84  {
85  }
86
87  void SMachine_s_2()
88  {
89  }
90
91  void SMachine_s_3()
92  {
93  }
```

```
94
95
96  //conditions functions
97  int SMachine_c_1()
98  {
99      if(func())
100       return 0;
101
102     return 1;
103 }
104
105 int SMachine_c_2()
106 {
107    return fun2();
108 }
109
110 int SMachine_c_3()
111 {
112     return fun3();
113 }
114
115 // transition functions
116 void SMachine_t_1()
117 {
118     do_something();
119 }
120
121 void SMachine_t_2()
122 {
123     do_something_else();
124 }
125
126 void SMachine_t_3()
127 {
128     do_something_completely_else();
129 }
```

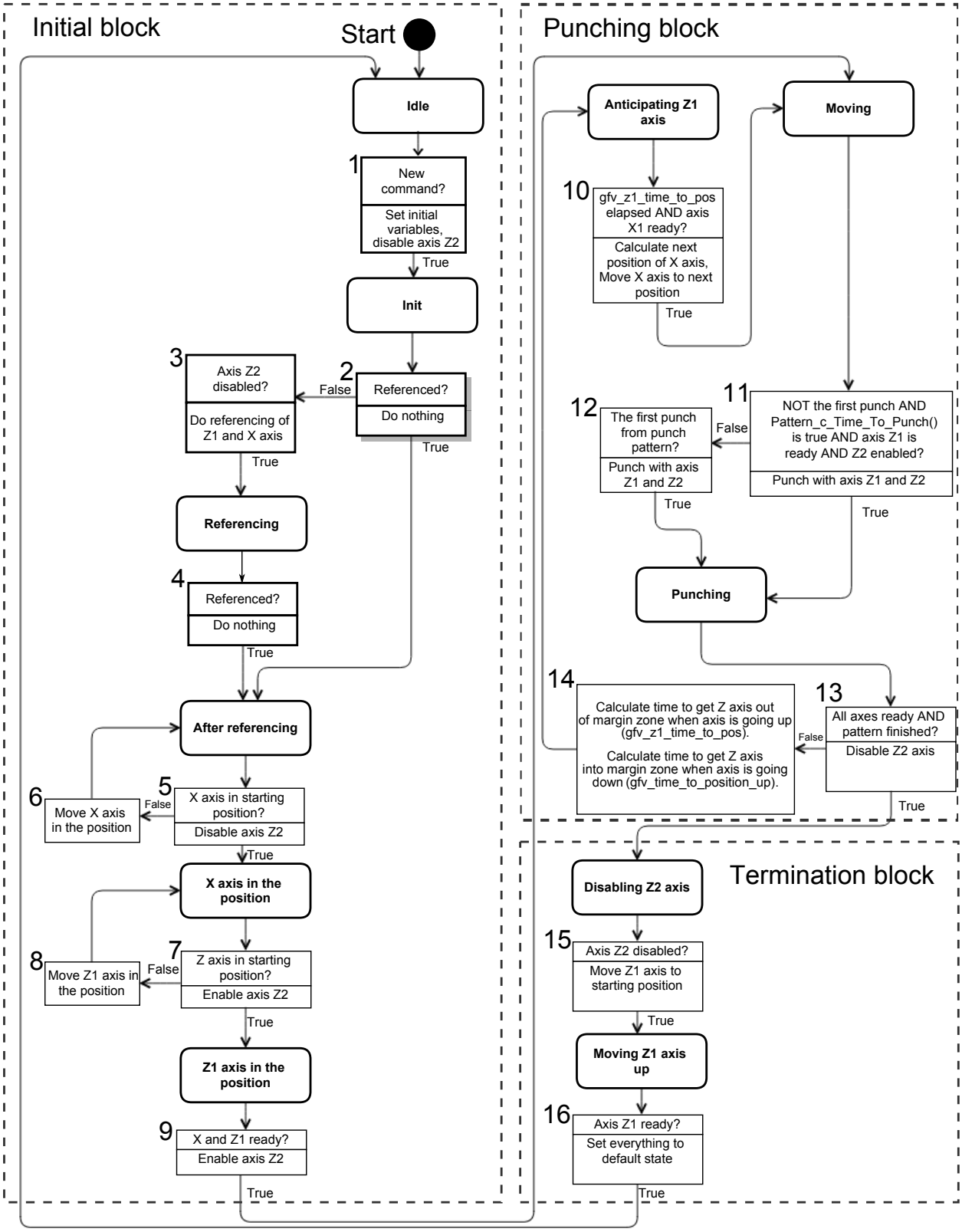


Figure 19: State machine for Dynapunch

5.2 Test pattern

The test pattern for testing Dynapunch performance is described in the following chapter. Pattern can have variable number of strokes, variable distance between strokes and also variable motion parameters for both axes.

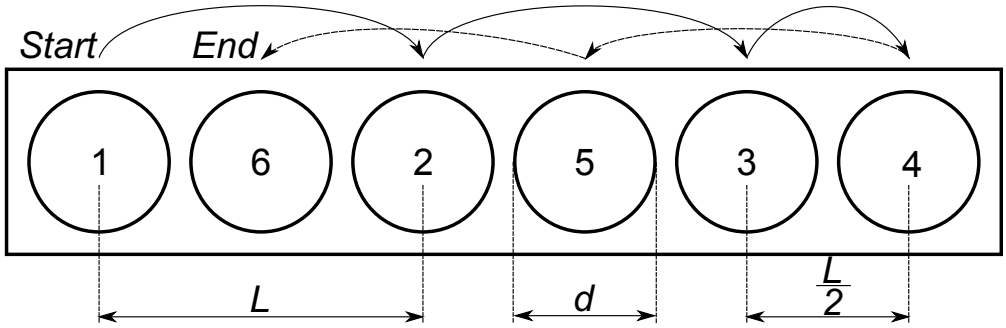


Figure 20: The test pattern

Holes are placed with distance L and respectively to numbers in figure 20. The diameter of the hole is d [mm], than $L = 2d+5$ [mm]. It leaves 2,5 mm gap between holes. The fourth hole is punched in the distance $L/2$ from the third hole. The direction of X axis movement is reversed after fourth hole. Holes 5 and 6 are perforated in between existing ones.

The pattern multiplies potential errors in axes synchronization. If axes are not synchronized well, distance between holes is not the same. The error is the most significant between holes 1, 6 and 2.

The motion profile of the punch pattern can be seen in figure 21. At first the X axis moves from its reference position to initial position. When the movement is finished, the Z axis travels from its referencing position to hover height. The punch pattern is started afterwards. The Z axis returns to the reference position after the pattern is complete.

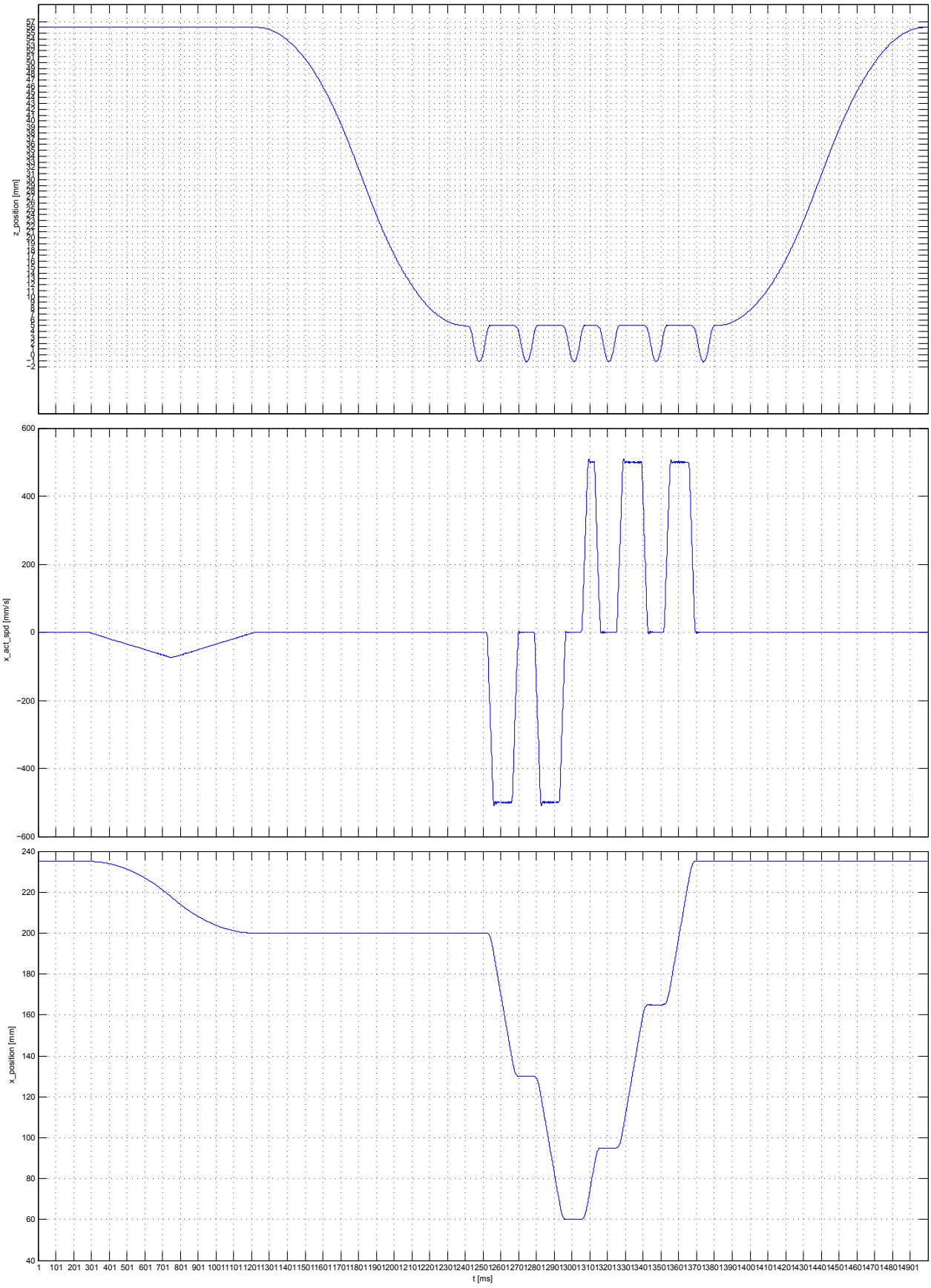


Figure 21: The punch pattern, the X and Z axis overview

6 Motion anticipation

The punch cycle requires well coordinated motion of the X and Z axis. As seen in figure 22, the ram is allowed to hit the workpiece in section 3 when the X axis is not moving. The ram travels through the workpiece. Travelled distance is called *clear height*. Then it continues under the workpiece into the die to reach desired *die penetration*. There is a space above clear height called *margin*.

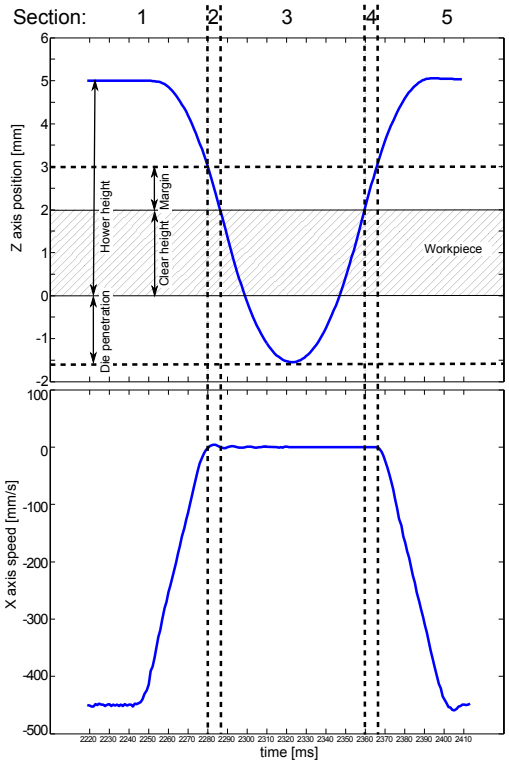


Figure 22: The punch cycle

If the X axis is moving, the ram is not allowed to be lower than the margin. It prevents collision of axes caused by tolerances in control process or uneven thickness of the sheet of metal. The margin is formed in sections 2 and 4 where the X axis is at a standstill and the Z axis enters or exits the margin zone. The X axis is changing position in zones 1 and 5 while the Z axis is reaching or leaving hover height. It is important to keep sufficient hover height during movement of the workpiece, because the workpiece might flicker on the table if a big sheet of metal was processed with high feed rate.

It is possible to wait until the X axis is completely in at a standstill and then starts the

movement of the Z axis. This approach is very simple, however it reduces the number of strokes per minute (SPM) rapidly.

The solution is to start movement of the Z axis before the X axis reaches its position and start the movement of the X axis right after the Z axis exits the safety margin zone. The thickness of the workpiece is not the same all the time, also hover height, die penetration and all other parameters such as speed, acceleration and jerk for both axes could differ from application to application. It is necessary to anticipate the position of axes. This approach is implemented into the state machine described in previous chapter.

There is a condition `Pattern_c_Time_To_Punch()` marked by number 11 in the state machine diagram (figure 19). The source code is in the listing 5. The condition has to be fulfilled before the Z axis starts the stroke. The condition compares two time variables.

Listing 5: Condition 11

```
1  int Pattern_c_Time_To_Punch()
2  {
3      float pfv_time_to_move_ready = axis_get_time_to_move_ready(X1_AXIS);
4      float pfv_time_to_pos = gfv_z1_time_to_pos+gfv_time_to_position_down_correction;
5      if((pfv_time_to_move_ready <= pfv_time_to_pos))
6      {
7        return 1;
8      }
9      return 0;
10 }
```

The first variable is the time to finish the movement of the X axis, the second one is the time to move the Z axis from the hover height to the margin zone. It was simple to get time for the X axis, because the function for it has already been implemented - `axis_get_time_to_move_ready(X1_AXIS)`. Variable `gfv_z1_time_to_pos` which stores the time for the Z axis is obtained by calling function `axis_get_time_to_position(Z1_AXIS, pfv_position)` in the transition marked as number 14 in the state machine diagram. The position to which the time is being calculated is called `pfv_position`. Source code of the transition is in the listing 6.

Listing 6: Transition 14

```
1  void Pattern_t_PunchToAnticipating_Z1()
2  {
3      debug_message(DEBUG_TAG,"Pattern_t_PunchToAnticipating_Z1");
4      int pfv_position = gfv_z1_axis_start_pos - gfv_z1_clear_height;
5      gfv_z1_time_to_pos = axis_get_time_to_position(Z1_AXIS,pfv_position);
6      gfv_time_to_position_up = axis_get_time_to_position_up(Z1_AXIS,pfv_position);
7  }
```

Function `axis_get_time_to_position(Z1_AXIS, pfv_position)` triggers a set of calls. The set of calls ends with the call of function `ecsrv_get_time_to_position(liv_ecsrv_idx,pfv_position)`, which actually does the calculation. The function can be viewed in the appendix D.2.

The function calculates end time of each motion zone (lines 9 - 133 in appendix D.2), the same as during the initialization of the trajectory generator in the chapter 4.2. Then `pfv_position` is compared (lines 165 - 181) with the end positions of time zones to detect which zone `pfv_position` belongs to. If the end zone is found, the time between the start of the end zone to the `pfv_position` is calculated (lines 165 - 278). Calculation is done by solving motion equations.

Motion equations are polynomial functions up to the 3rd older in the form $p = at^3 + bt^2 + ct + d$ where a, b, c, d are real numbers, t [s] is an unknown variable and s [mm] is `pfv_position`. Calculation of t is done between lines 179 - 266. It is simple to get t in zones t1 and t4 where polynomial has form $p = at^n + d$. It can be solved directly. Also zones t2 and t6 have a simple solution, because the polynomial degree is 2. The quadratic formula $t_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ is used.

The solution in zones t3 and t5 is more difficult. The polynomial degree is 3, usage of quadratic formula or direct solution is not possible. The speed of CPU does not allow to use brute force - the cyclical increase of t with small step and comparison of results with `pfv_position`, it would be too slow. It is a good idea to use the Newton–Raphson method. An example of usage for the zone t3 is in the listing 7. Polynomial coefficients a, b, c, d are saved in the field `c[3]` and forwarded to the function `tgen_newton(float c[3], int power, float start, float max, float min)`. The function is described in the code listing 8. The function `tgen_newton` has its precision interval set to 0.0001 mm. The maximum number of iteration is set to 20.

Listing 7: Usage of the Newton–Raphson method in the zone t3.

```
1  case 3: // zone t3
2       {
3         float c[3] = {0};
4         c[0] = s2 - pfv_position;
5         c[1] = v2;
6         c[2] = a/2.0;
7         c[3] = -j/6.0;
8         pfv_time_B = tgen_newton(c,3,0.0001,t3,t2);
9         pfv_pos_B = s2 + v2*pfv_time_B + a*pfv_time_B*pfv_time_B/2.0 - j*pfv_time_B*pfv_time_B*pfv_time_B/6.0;
10        break;
```

```
11      }
```

Listing 8: Newton–Raphson method

```
1  float tgen_newton(float c[3], int power, float start, float max, float min)
2  {
3  // Newton-Raphson method
4  // c[n] - coefficient of n-th order
5  // power - order of polynomial
6  // start - starting point
7  // max, min - interval borders
8
9      int i=0,count=0;
10     float c[5]={0};
11     float x1=start,x2=0,t=0;
12     float fcn1=0,fcn1_derived=0;
13
14     do {
15             count++;
16             fcn1=fcn1_derived=0;
17             for(i=power;i>=1;i--)
18             {
19                 fcn1+=c[i] * (pow(x1,i)) ;
20             }
21             fcn1+=c[0];
22             for(i=power;i>=0;i--)
23             {
24                 fcn1_drived+=c[i]* (i*pow(x1,(i-1)));
25             }
26             t=x2;
27             x2=(x1-(fcn1/fcn1_drived));
28
29             x1=x2;
30      } while(((fabs(t - x1))>=0.0001) & (count<=20));
31     return x2;
32 }
```

When the stroke is triggered, the state machine goes to the state *Anticipating of Z1 axis* and waits until condition `Pattern_c_Time_To_Move()` in transition 10 is fulfilled. The condition is listed in the code listing 9. If the condition is fulfilled, the X axis moves to the next position. The condition compares actual time spent during movement of the Z axis with pre calculated time `gfv_time_to_position_up`. It is the time to reach end of the margin zone when the Z axis is returning to the hover height. Time `gfv_time_to_position_up` is calculated in the same way as time `gfv_z1_time_to_pos`.

Listing 9: Condition 10

```
1  int Pattern_c_Time_To_Move()
2  {
3      float pfv_act_time = axis_get_act_time(Z1_AXIS);
4      if((pfv_act_time >= gfv_time_to_position_up+gfv_time_to_position_up_correction))
5      {
6        return 1;
7      }
8      return 0;
9  }
```

Correction factor `gfv_time_to_position_up_correction` is added to the `gfv_time_to_position_up` in the condition 10. Similar factor `gfv_time_to_position_correction` is added to the `gfv_time_to_position` in the condition 11. It is necessary to compensate reaction times of drives and transport delays. It is discussed in following chapter.

6.1 Axis synchronization

As you can see in condition 10 (listing 9) and in condition 11 (listing 5), the correction factor is added to calculated times. It is necessary to do it, because there are transport delays of the signal and response times of drives.

It takes two cycle times (2 ms) to send the command to the drive. The drive position control loop is running on another frequency than real time PC. Cubic interpolation is used to resample command signal to frequency used in drives. Sampling adds approximately 2 ms of transport delay. Another delay is caused by signal processing inside drives. After that drives start the commanded action. All this gives a total delay of 6 ms.

After the drive starts the action, there is another delay until the motors are really rotating and an information about position is back to the real time PC. It gives approximately two more cycle times of delay. The situation is depicted in figure 23.

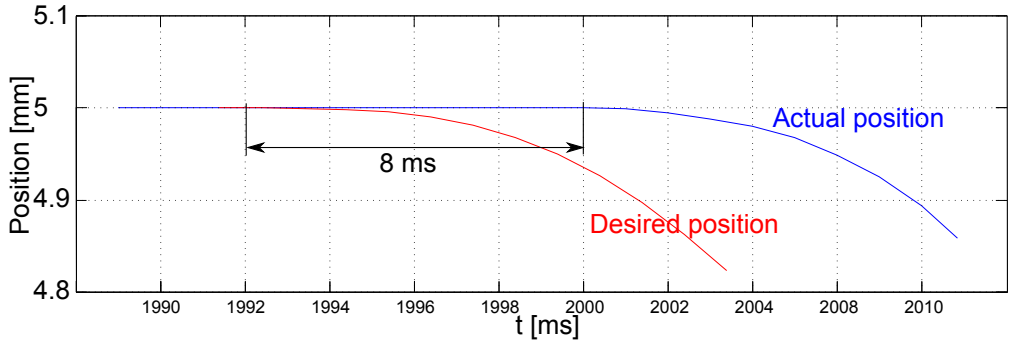


Figure 23: Delay between desired and actual position

It is important to know that delays are cancelled if times for desired positions of the X and Z axis are compared. In the case of `gfv_time_to_position_correction` desired value for the Z axis is compared with the desired position of the X axis. It means that 8 ms delay is cancelled and only the delay dependant on the structure of state machine and deviation caused by sampling remains. It is the same situation with `gfv_time_to_position_up_correction`,

but axes are commanded in reversed older. It is the reason for the negative sign of `gfv_time_to_position_up_correction`.

Correction factors in the conditions 10 and 11 are adjusted to:

`gfv_time_to_position_up_correction` = −0,003 [s]

`gfv_time_to_position_correction` = 0,002 [s]

Testing with various parameters of motion profile proved that correction factors do not depend on the motion profile. There is a testing plot in figure 24. It consist of three graphs. The first one contains the desired position of the Z axis (red line) and the real position of the Z axis (blue line). The second graph shows the actual and desired speed of the X axis. Colouring of lines is the same as in the first graph. The position of the X axis can be seen in the last graph.

A closer look at figure 24 shows that the margin zone is set to 2,5 mm. The Z axis reaches this point at the same time as the X axis reaches zero speed. There is a small overshoot, but position change is less than 0,2 mm which can be neglected because the sheet is penetrated after the biggest peek. When the Z axis goes up, the X axis is at a standstill until the Z axis steps out of the margin zone. Other testing plots with different hover height, die penetration, distance between strokes or punching speed are listed in the appendix B.

The plotting tool for investigation of motion graphs was created. It is written in Matlab programming language. It is able to plot graphs from raw data provided by the real time PC. It is possible to pick the particular time interval from the motion profile and it is also possible to define which motion properties such as acceleration, speed or actual current will be plotted. Plots are aligned vertically on top of each other with a common x axis which represents time. The result plot is exported to PDF automatically. Source code can be found in the folder *Plotting tool* in the attached CD (attachment A).

6.2 Summary

The method for motion anticipation is developed and tested. SPM ratio is improved, because the axes do not have to wait for each other. The method is independent of settings of axes parameters. It does not require a lot of CPU time, because motion equations are not solved by brute force.

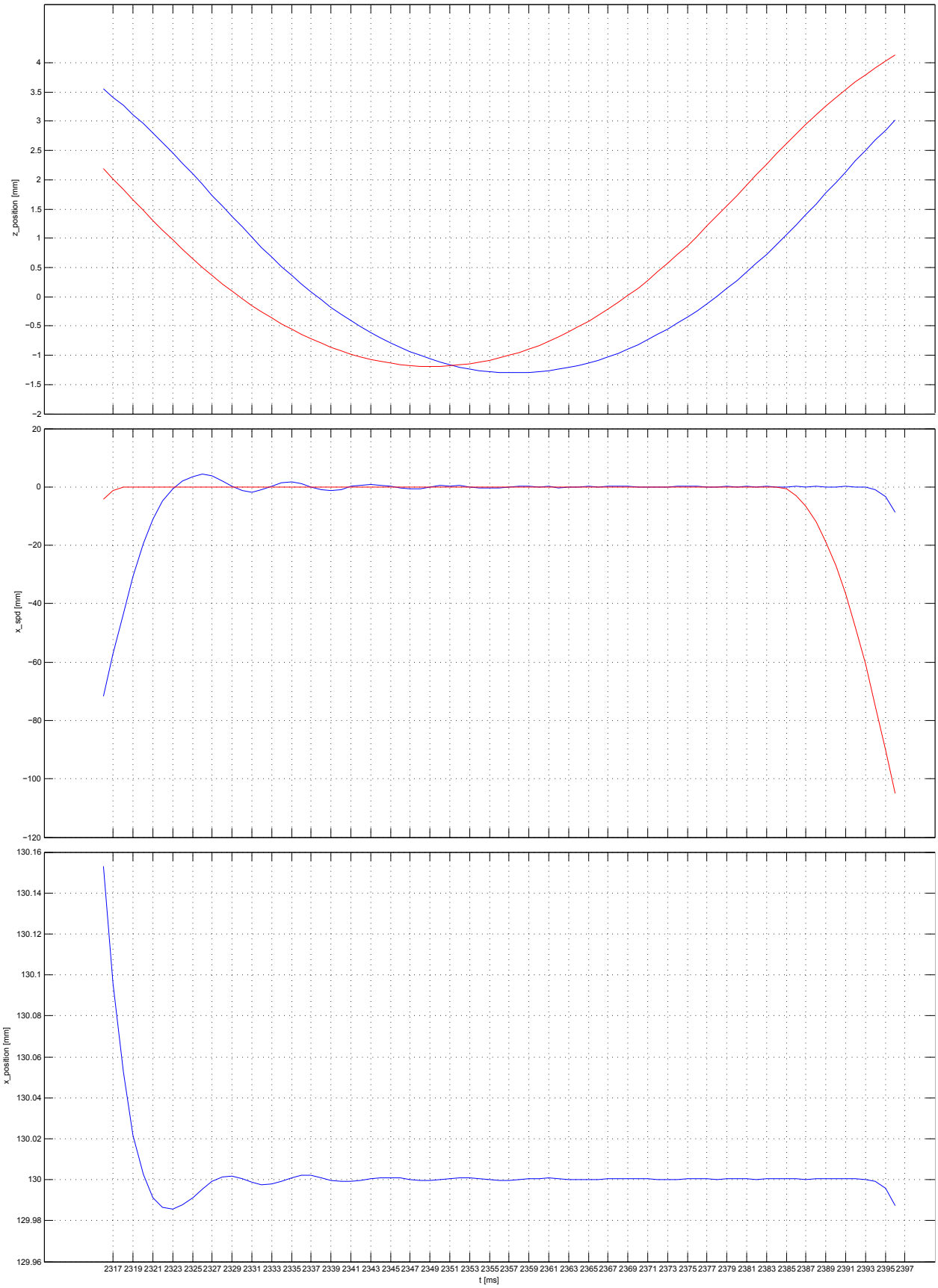


Figure 24: Axes synchronization test, Parameters of the Z axis: hover height $h = 2.5$ mm, die penetration $d_p = 1.2$ [mm], jerk $j = 1.041$ ms^{-3}, max. acceleration $a_{max} = 62.5$ ms^{-2}, max. speed $v_{max} = 0.179$ ms^{-1}. Parameters of the X axis: stroke distance $s_d = 70$ mm, jerk $j = 3500$ [ms^{-3}], max. acceleration $a_{max} = 35$ ms^{-2}, max. speed $v_{max} = 0.7$ ms^{-1}

7 Two motors

The Z axis is powered by two identical motors interconnected by constant ratio gearbox. Each motor has its own drive. There is no possibility to command the first drive and copy its motion directly to the second one, because drives do not have implemented any direct drive to drive communication. The only option is to command motors simultaneously from the real time PC.

It is also not possible to send identical position commands into both drives. There is a backlash between teeth in the gearbox as it is depicted in figure 25. If motors were position commanded, one would propel the main wheel in the gearbox and the other would follow with a backslash without even touching the main gear.

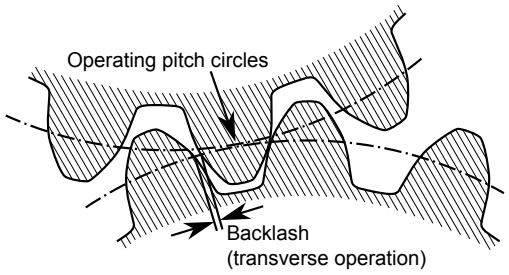


Figure 25: Gear backslash, source in the foot note 1.

There is also a possibility that motors are unevenly synchronized. If one motor was trying to reach the desired position, the other would already be in the position and push back against the first motor. It would cause arguing between motors, which would lead to oscillations and premature wearing of shafts and the main gear.

Figure 26 shows the detailed scheme of the control loops inside the drive. Blocks *Ft*, *Fv* and *Fp* are constants to convert values to ranges used in the drive. There are blocks marked as *Filter*. They are low pass filters to suppress higher frequencies in the control scheme. It is possible to send position commands to the drive and apply torque and velocity offsets.

7.1 Torque synchronization

The first option is to send position commands to the first drive (noted as Z1), read the value of actual current (G25P08 in the scheme) generated by the command and apply current

[1] `http://commons.wikimedia.org/wiki/File:Backlash.svg#mediaviewer/File:Backlash.svg`

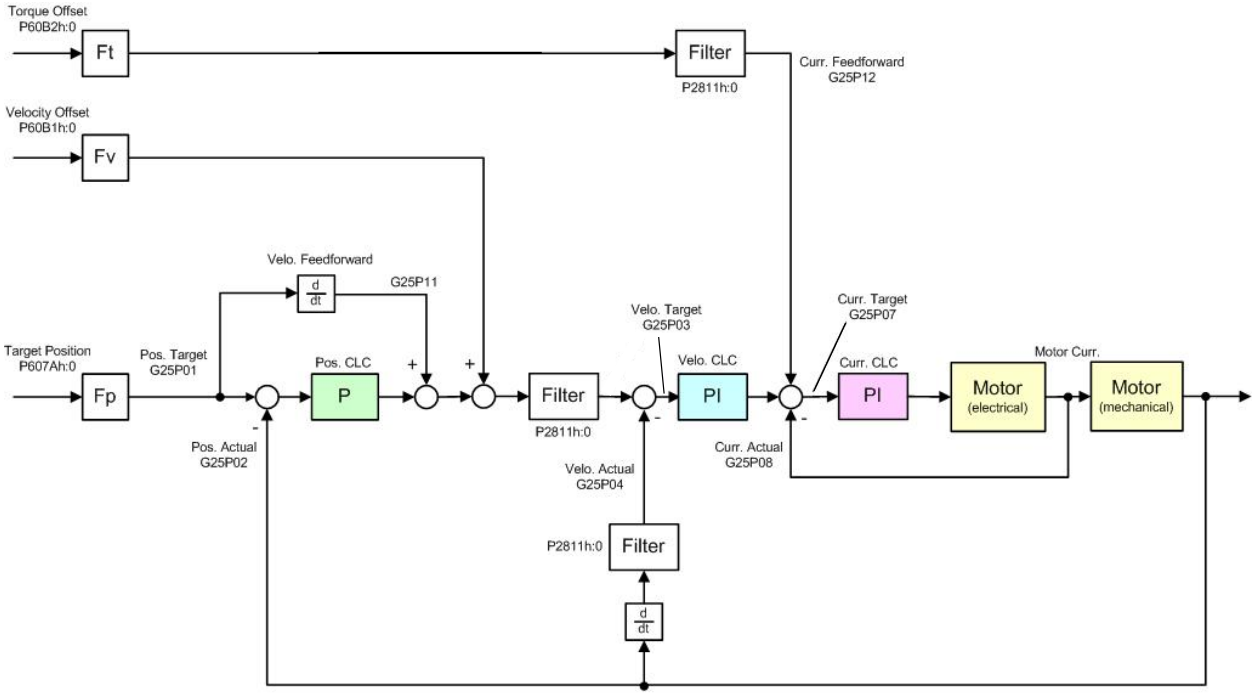


Figure 26: Detail scheme of control loops

offset to the second drive (noted as Z2). However, due to transport delays between drives and the real time PC it is not possible to synchronize motors properly. As seen in figure 27, current of the Z2 drive lags behind Z1. There would be no section with current values around 0 A in the case of faster movement. Motors would attempt to turn in the opposite direction.

7.2 Velocity synchronization

The second option is to control both motors by velocity commands instead of position commands. It is possible to do it through the input called velocity offset. Proportional gain of the position control loop inside drives is set to zero and feed forward is disconnected. The position control loop is placed out of drives to the real time PC. Output of the loop is sent to both drives. The value of actual position is taken from the Z1 drive and used for both drives. Position of the Z2 motor is not important, there is only one requirement - the velocity of Z2 motor must be the same as velocity of the Z1 motor. The whole control scheme is in figure 28.

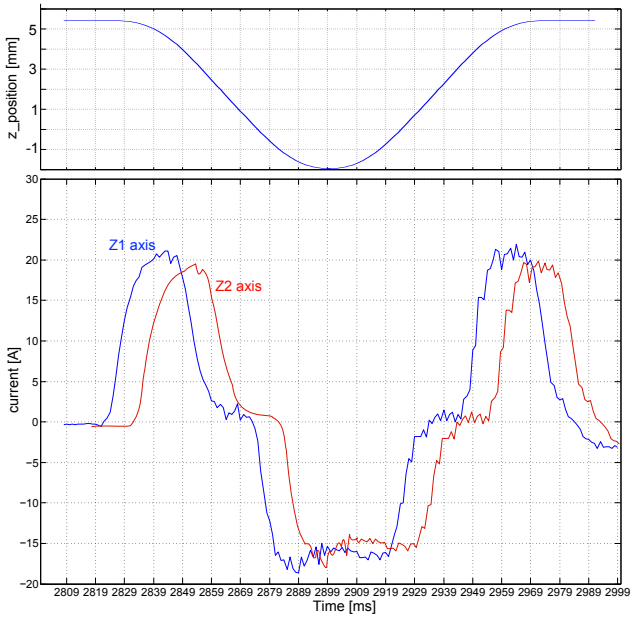


Figure 27: Incorrect torque synchronization of drives

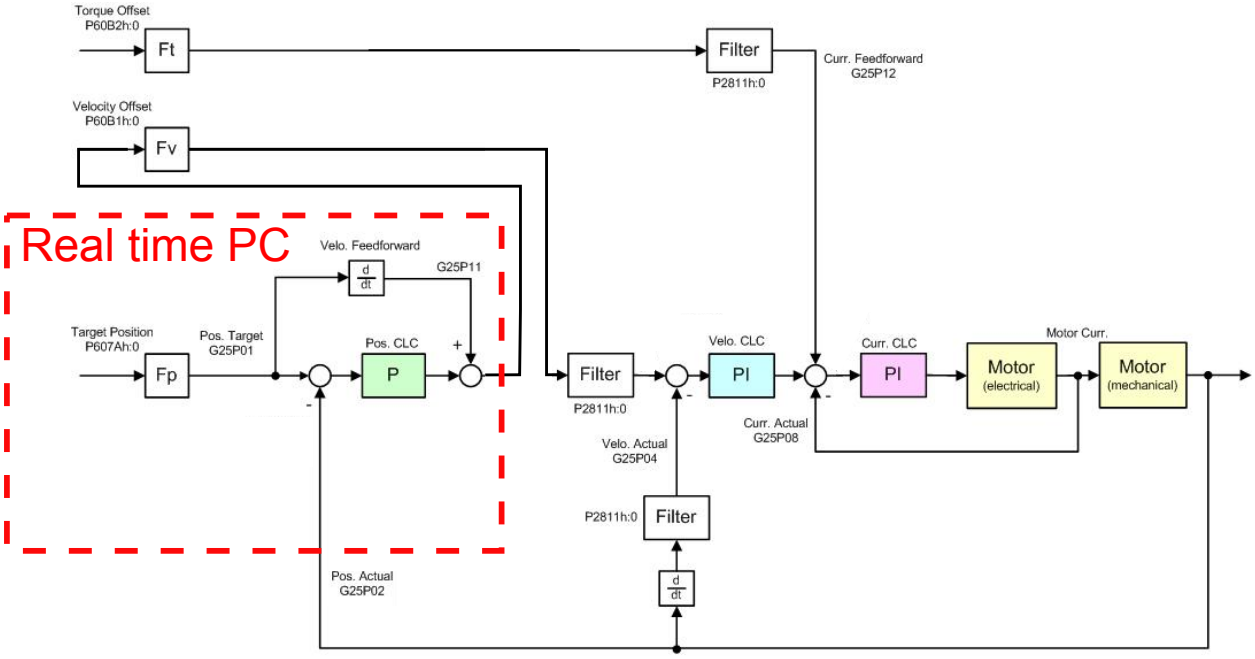


Figure 28: Position loop outside the drive

The position control loop placed in the real time PC is implemented by the function `position_loop`. Source code of the function is in the code listing 10. It is executed every 1 ms, which is enough for good position regulation. Line 4 restricts usage of external position

control loop only for the Z1 and Z2 axes. Lines 9 - 17 corrects proportional gain in low speeds for better performance. This method is called gain scheduling. Regulation itself is located in lines 24 and 25. The result is `req_speed` which is recalculated to `req_speed_inc` - velocity in increments per second. Values used for recalculation are: gearbox ratio 21/133, spindle ratio 1/20 mm and the number of increments per revolution 65536.

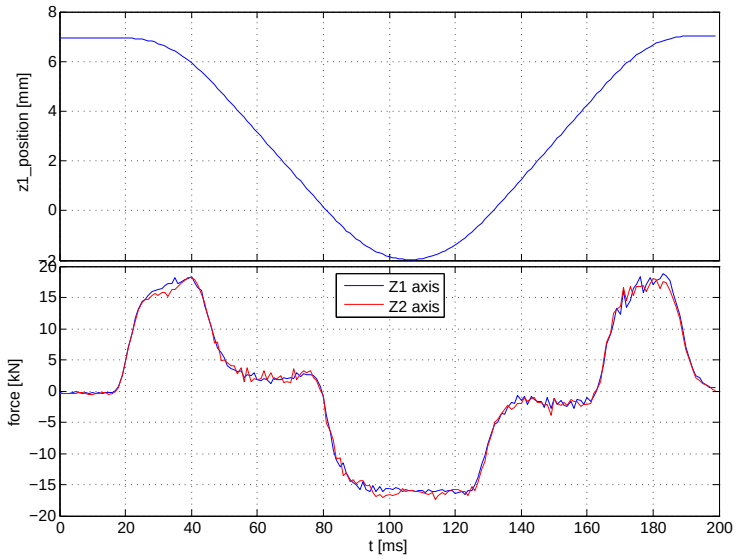


Figure 29: Synchronization of drives

Motors are well synchronised as seen in figure 29. The force in the bottom plot is proportional to current by the factor k_t mentioned in the chapter 3.4. Force curves differ slightly, but this is natural. Motors and drives are not identical copies of each other.

Listing 10: Position control loop

```
1  float i_part[5] = {0};
2  void position_loop(ttv_ecsrv_control* cnt_poi)
3  {
4       if( (cnt_poi->axis_kind == 2) || (cnt_poi->axis_kind == 3))
5       {
6            float pos_p_gain = cnt_poi->init_data.sfv_positions[5];
7
8            // increase proportional gain in low speeds - gain scheduling
9            if(cnt_poi->move_data.siv_punch_cycle == 8)
10           {
11             if(cnt_poi->act_spd_mm < 10)
12             {
13               pos_p_gain = 3 * pos_p_gain;
14               if(pos_p_gain < 2000)
15                 pos_p_gain = 2000;
16             }
17           }
18
```

```
19            if( pos_p_gain > 4000) // max. allowed value
20              pos_p_gain = 4000;
21
22            float pos_ff_velo_gain = cnt_poi->init_data.sfv_positions[6];;
23
24            float pos_diff = cnt_poi->req_pos_mm - cnt_poi->position_mm;
25            float p_part = pos_diff * pos_p_gain * pos_p_gain_factor;
26
27            float velo_feedforward = pos_ff_velo_gain * cnt_poi->req_spd_mm; // feed-forward is calculated from
                    required speed
28
29            float req_speed = ( p_part + velo_feedforward ); // [mm/s]
30            int req_speed_inc = (int) ((( -1 * req_speed * 133 * 65535) / 20.0) / 21.0); // conversion to
                   increments
31
32            cnt_poi->siv_vel_offset = req_speed_inc;
33
34            // to store required speed of Z1 axis
35            if(cnt_poi->axis_kind == 2)
36            {
37              giv_req_z1_speed = req_speed_inc;
38            }
39      }
40  }
```

7.3 Summary

Two methods for motors synchronization of the Z axis are tested. Torque synchronization turned out to be impracticable due to transport delays between the real time PC and drives. The velocity synchronization method is successful although it is necessary to transfer the position control loop outside drives into the real time PC.

8 Drives tuning

Tuning of drives can be achieved by many approaches. Methods can be divided into two main groups. The first group represents on-line tuning methods and the second one off-line tuning methods.

8.1 Manual tuning of the Z axis

Manual tuning belongs in the group of on-line methods. It requires some level of experience and in some cases many attempts to reach good results. The advantage of manual tuning is that it requires small or no knowledge of the internal structure of the controller. Manual tuning is the easiest method of tuning controllers. Control loop gains are changed until a satisfying result is obtained. Table 2 shows how the system responds if single value of control loop gain is increased.

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
k_p	Decrease	Increase	Small change	Decrease	Degrade
k_i	Decrease	Increase	Increase	Decrease significantly	Degrade
k_d	Minor decrease	Minor decrease	Minor decrease	No effect in theory	Improve if Td small

Table 2: Effect of increased control loop gain

According to instructions in the IRT Instruction manual [9], the following tuning approach can be used:

- Set all gains to 0
- Send step current with maximum motor current value
- Set proportional gain k_p according to figure 30
- Set proportional gain k_i according to figure 31

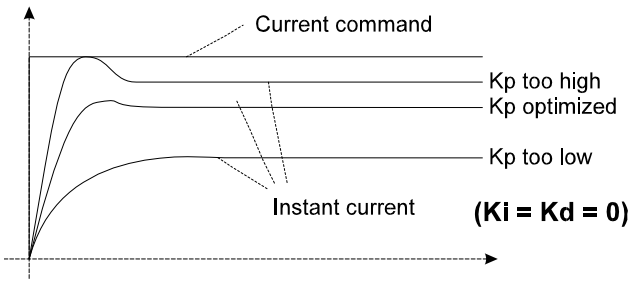


Figure 30: Current loop tuning k_p, source [9]

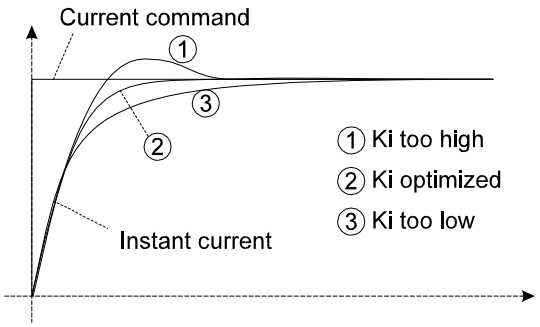


Figure 31: Current loop tuning k_i, source [9]

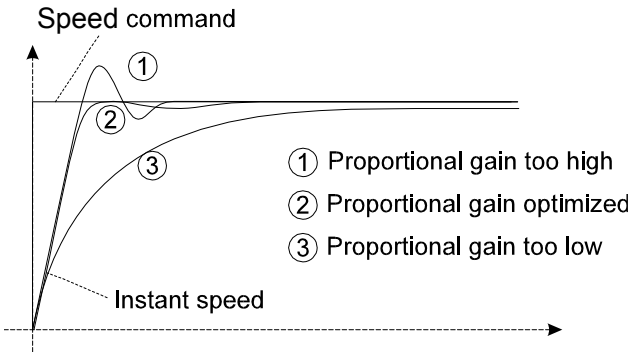


Figure 32: Speed loop tuning k_p, source [9]

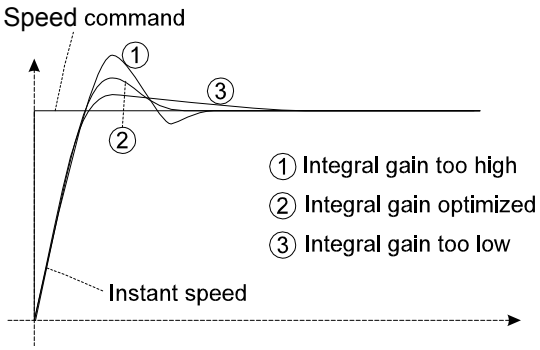


Figure 33: Speed loop tuning k_i, source [9]

The same approach can be used for tuning speed loop, it is depicted in figures 32 and 33. Cyclically repeating current steps are applied into the drive through *Torque offset* input. Steps have amplitude from -20 A to 20 A with a period of 200 ms.

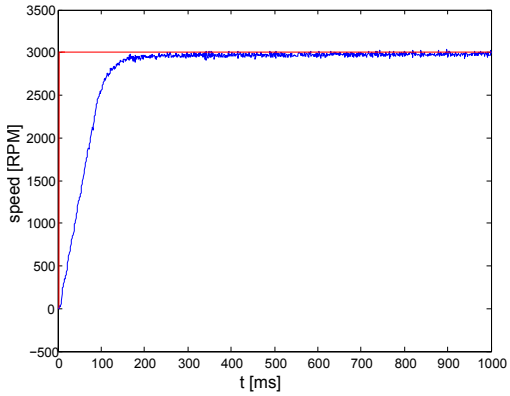


Figure 34: Speed loop tuning big step

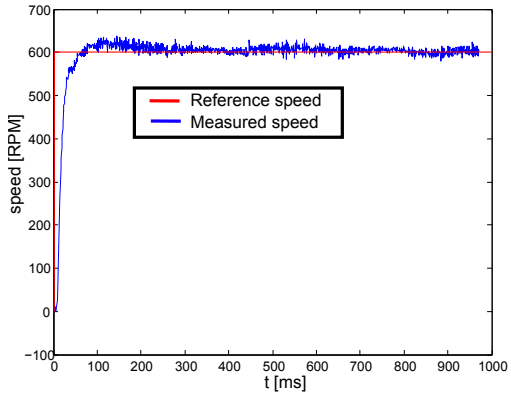


Figure 35: Speed loop tuning small step

After a few iterations, optimal setting was found. Parameters are listed in the table 5. Current follows reference variable without overshoot and with a short rise time. Response to step change of current is depicted in figure 36. Control loop gains are so high, because

Position loop	k_p
1	100
2	200
3	400
4	1000
5	2000
6	800
7	700
8	750
9	850
10	800

Table 3: Tuning position loop of Z1 axis

Speed loop	k_p	k_i
1	10000	0
2	20000	0
3	40000	0
4	60000	0
5	50000	0
6	40000	0
7	39000	300
8	39000	200
9	39000	100
10	39000	10
11	39000	50
12	39000	25
13	40000	25
14	40000	20

Table 4: Tuning speed loop gains of Z1 axis

Current loop	k_p	k_i
1	2000	0
2	1000	0
3	500	0
4	600	100
5	600	150
6	700	150
7	750	150
8	750	140

Table 5: Tuning current loop gains of Z1 axis

	k_p	k_i
Position loop	$10^{-5}\ \frac{RPM}{increment}$	-
Speed loop	$10^{-6}\ \frac{A}{RPM}$	$10^{-2}\ \frac{A}{RPM \cdot s^{-1}}$
Current loop	$0.005\ \frac{\%}{A}$	$40\ \frac{\%}{A \cdot s^{-1}}$

Table 6: Conversion factors

they are multiplied by conversion factors inside the drive. Conversion factors are listed in the table 6.

The same method is applied to the speed control loop. Tuning is done on two velocity steps. The first one goes from 0 RPM to 3000 RPM and the second one goes from 0 RPM to 600 RPM. Two steps help to validate tuned parameters for low and high speeds. Tuning parameters are listed in the table 4. Step responses are in figures 34 and 35.

The position control loop consists of proportional controller with feed-forward. It is placed in the real time PC and executed every 1 ms. The tuning sequence is in the table 3. Tuning is done on punching profile, tuning on step position change does not give good results, the motion profile described in chapter 4 is used. Graphs describing results of tuning are in

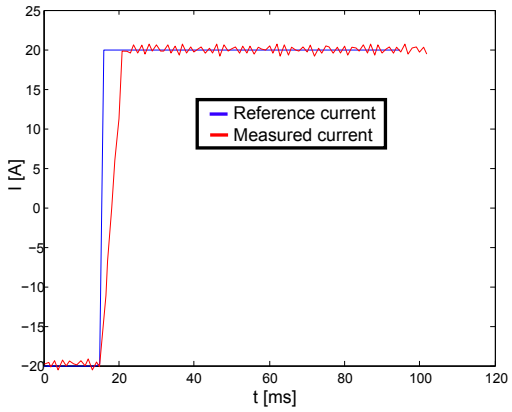


Figure 36: Z axis step response of current loop

the appendix C. The first plot is in figure 76. Red lines in the first and second plots mean reference variable and blue lines mean measured variable. The red line in the third (force) plot is the force produced by the Z1 motor and blue stands for force produced by the Z2 motor. The tuned setting has a small overshoot at the end of the movement. Settling time more than 200 ms is longer than it should be. The gain scheduling is used to improve the shape of the position curve. Properties of the gain scheduling can be seen in the code listing 10 in the line 9. Motion profile with gain scheduling is in figure 77. Overshoot is reduced and settling time is less than 20 ms.

A slow punch through 5 mm thick steel plate is depicted in figure 78. The force plot shows that the press force required for perforation of the plate is $F_p = 2 \times 85 = 170$ kN. It can be compared with a faster punch through the steel plate with the same thickness in figure 79. Although the press force is nearly the same, the peak lasts 10 ms. Peak duration of the slower punch is 15 ms. It is caused by inertial force, which is bigger with the higher impact speed.

Finally, the punch patterns with and without steel plate were tested. The results are in figures 80 and 81. Press force is lower compared with previous plots, because a different die is used. Force plot shows that acceleration requires more force than perforation of the sheet. Peaks in the second and third stroke were caused by the misalignment of the tool which was hitting the rim of the die. The tool had to be replaced after that.

8.2 Manual tuning of the X axis

Tuning of the X axis has been done in the same way as tuning of the Z axis. There are no points of interest to describe except the requirement for accuracy of positioning. The X axis should be able to hold position with tolerance ± 0,05 mm. The result parameters are listed in the table 7. Detail of tuned response can be seen in figure 37. The motion is done with maximum speed set to 80 mm/s. Position error is less than 0.02 mm. Accuracy meets requirement and it remains in the desired tolerance with usage of higher or slower maximum speeds.

Param.	Position loop	Speed loop	Current loop
k_p	600	20010	8000
k_i	–	80	3000

Table 7: Tuned gains of the X axis

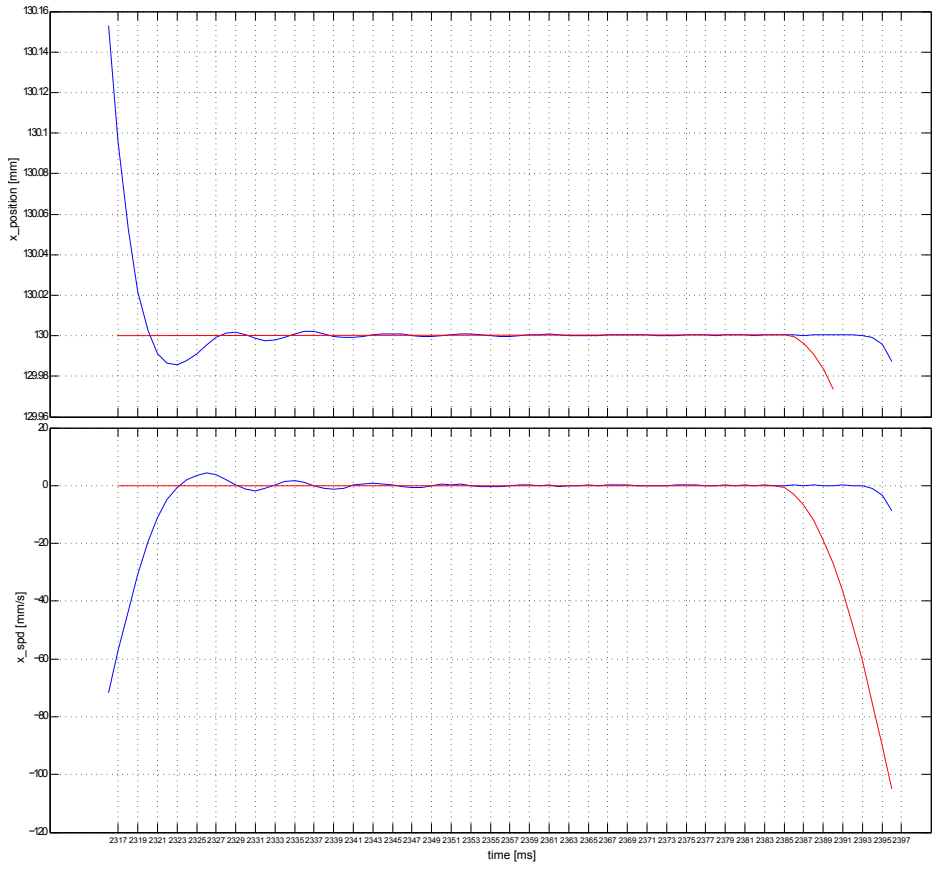


Figure 37: The X axis tuned

8.3 Hit rate

Hit rates or SPM ratio is highly dependant on the length of the stroke and also on the distance travelled by the X axis between holes.

The punch pattern in figure 81 is considered as a worst case. The stroke length is 9 mm and distance between hole centres is 35 mm. The stroke lasts 160 ms and the movement of the X axis adds 60 ms. The hit rate is 272 SPM. It is possible achieve 500 SPM if the stroke length is reduced to 5 mm. This hit rate is comparable with mid-class hydraulic punch presses. If the distance between holes is reduced to 1 mm, the hit rate is reaching 600 SPM.

8.4 System modelling

Control loop gains were tuned manually in the previous chapter. It would be convenient to create a model of the axis to perform tuning on it. If the model is precise enough, the time required for tuning will be shorter and also the result should be more precise. There is a risk that the model will have to be very complex for good approximation of the real plant. It would be better to use other approaches in that case. The Simulink environment is used for creation of the model.

Figure 38 shows the entire model of the Z axis. There is one motor connected to the axis. The block on the right side of the scheme represents a model of the DC motor. The block called "Axis" represents the rest of the mechanical part of the axis. Blocks on the left belong to the servo drive. There are position, speed and current controllers. It is possible to disconnect controllers and apply constant values instead of them.

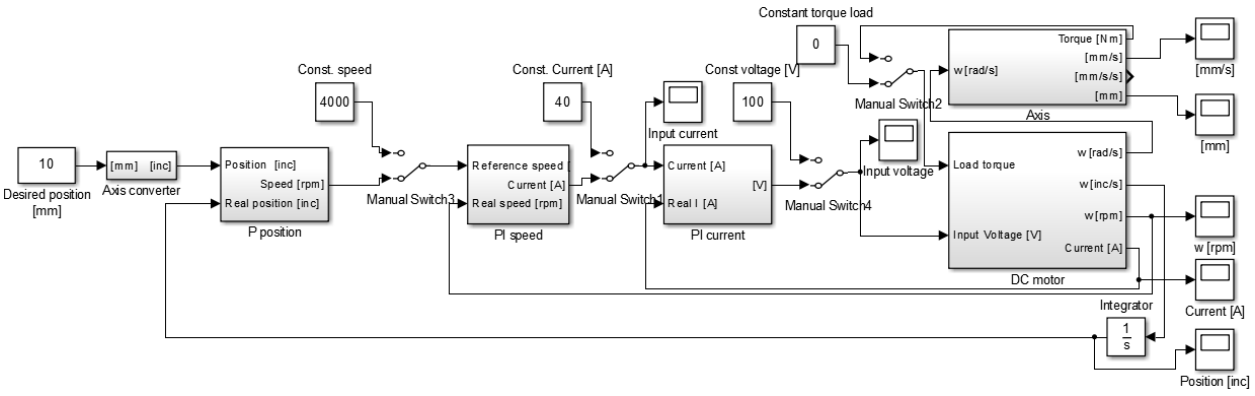


Figure 38: The model overview of the Z axis

The DC motor is simulated according to equations presented in chapter 3.4. The simulation scheme is depicted in figure 39.

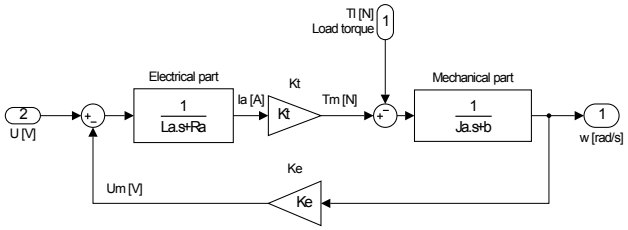


Figure 39: The model of the DC motor

Blocks representing controllers are in figures 40, 41 and 42.

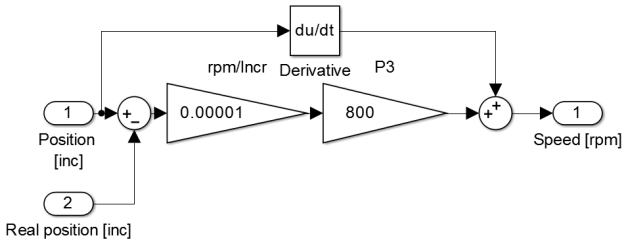


Figure 40: Position controller

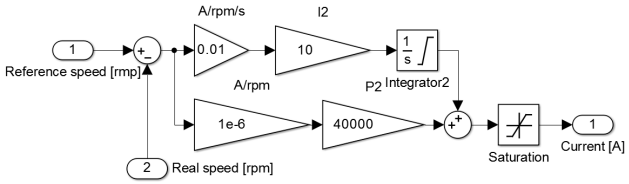


Figure 41: Speed controller

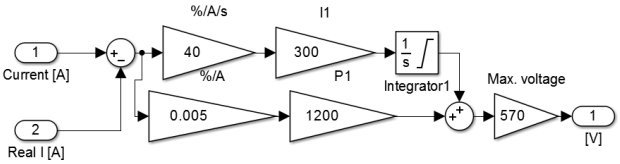


Figure 42: Current controller

The block "Axis converter" serves for converting the position of the axis [mm] to position of the motor [increments]. The block "Axis" serves for the conversion of angular units back to the transversal. Torque output of the block is not yet implemented. Function of the motor and the controller has to be verified first.

To prove that the model is a good approximation of the real system, the measurement of a standalone motor is made. The motor is disconnected from the rest of the machine and the response to step change of current is measured. Current is applied to the regulator through *Torque offset* input. The current regulator is set to values obtained by manual tuning. Step response is in figure 44.

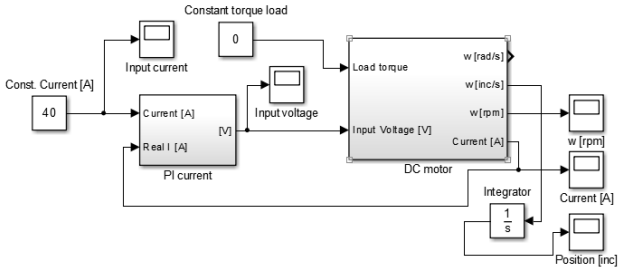


Figure 43: Simulation scheme - step response

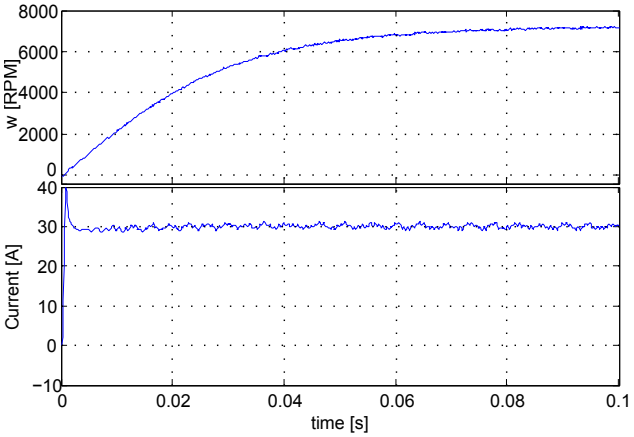


Figure 44: Step response of the real system

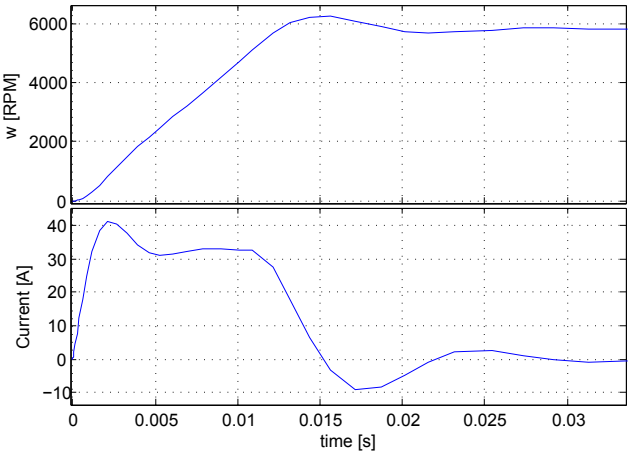


Figure 45: Step response of the model

Same steps, as described above, are done in simulation. Simulation scheme is depicted in figure 43. The result can be seen in figure 45. However the step response of the model does not match the response of the real system. Some similarities can be found, but it is far from a good match. The weak spot of the model is in the servo drive model. It is known that PWM voltage source provides maximum voltage 570 V. That value is multiplied by the percentage factor in the model. However no additional information about PWM voltage source is provided. Even if steady state values would be the same, it would be complicated to simulate small oscillation in the current curve. It is more convenient to simulate the system in another way. All models and data are in the attached CD (attachment A).

9 System identification

This chapter chapter describes the method for identification of the system model. System identification is based on mathematical analysis of input and output signals from the examined system. Knowledge of inner properties of the system is not necessary. The result of system identification is a mathematical model, in this case transfer function. Data is processed by MATLAB System Identification Toolbox.

9.1 System frequency response

Frequency response of the system is obtained by application of a wide band signal into the drive. Various types of excitation signal can be used, such as swept sine, multi-sine or white noise. Sine sweep is the best choice, because it has the simplest implementation. Measured output variable is velocity of the axis.

There is need to improve accuracy of measurement. A special type of sine sweep was developed. Sine sweep is characterised by amplitude, maximum frequency, minimal frequency and time period. Frequency of the sine sweep rises from minimal frequency to maximum and then decreases back to the minimum in the given time period. Each frequency is measured twice. Once if frequency is rising and again if frequency is falling. Amplitude of the signal is constant. It is depicted in figure 46.

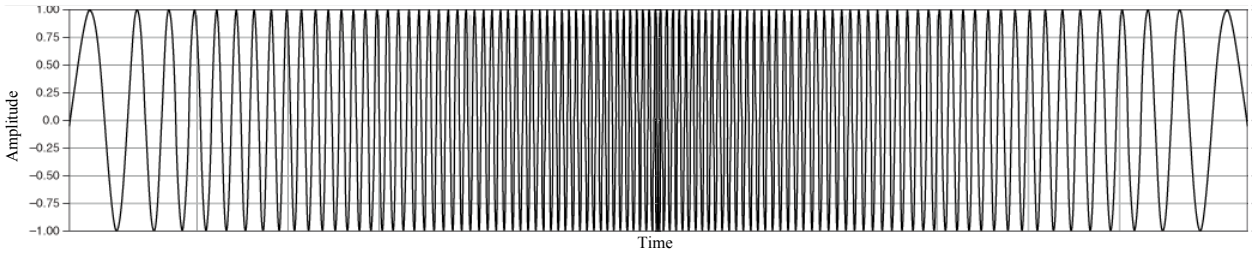


Figure 46: Sine sweep signal

It is necessary to measure frequencies from 0,1 Hz to 400 Hz with sampling frequency 1 kHz. The frequency corresponds with the period cycle time of the embedded PC. There are memory limitations which allow measurement only for a time period of 60 s. If the whole frequency range was measured in one 60 s block, there would only be a few samples for each frequency. The quality of measured data would be very low and it would be impossible

Part	Frequency range
1	0,1 Hz to 0,5 Hz
2	0,4 Hz to 0,1 Hz
3	0,9 Hz to 5 Hz
4	4 to 10 Hz
5	9 to 20 Hz
6	19 to 50 Hz
7	49 to 100 Hz
8	99 to 200 Hz
9	199 to 400 Hz

Table 8: Frequency ranges

to do proper system identification. Therefore the frequency range is divided into blocks according to the table 8. There is a frequency overlap on sides of neighbouring blocks for better continuity of measured data.

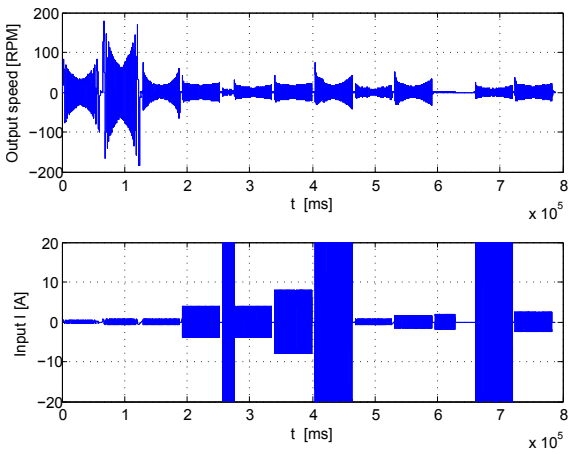


Figure 47: Sine sweep

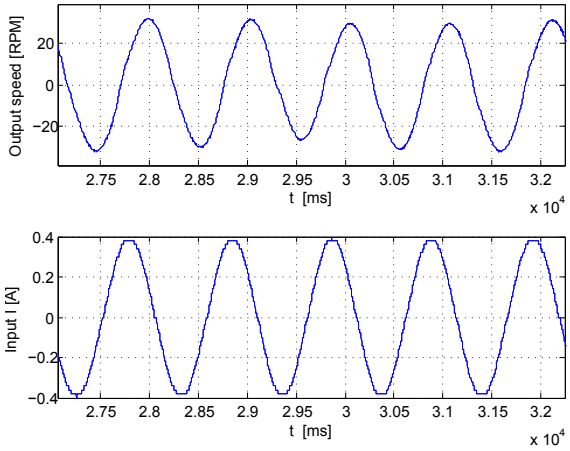


Figure 48: Detail of sine sweep

The reference signal, current I [A], is applied to the current regulator and speed of the axis is measured and recalculated to the speed of the motor ω [RPM]. Measured sweeps are in figure 47. Detail of the sweep is in figure 48.

All models and data are in the attached CD (attachment A). Measured data blocks are saved as a plain text in columns. It is necessary to convert them to the form processable by Matlab. Matlab script *load_scopes.m* is used for that. Script loads all measured blocks and connects them together. Then script *bodes_tfestimate.m* is launched. There is the function `tfestimate`, which estimates gain and phase shift between input and output data for frequencies in a certain range.

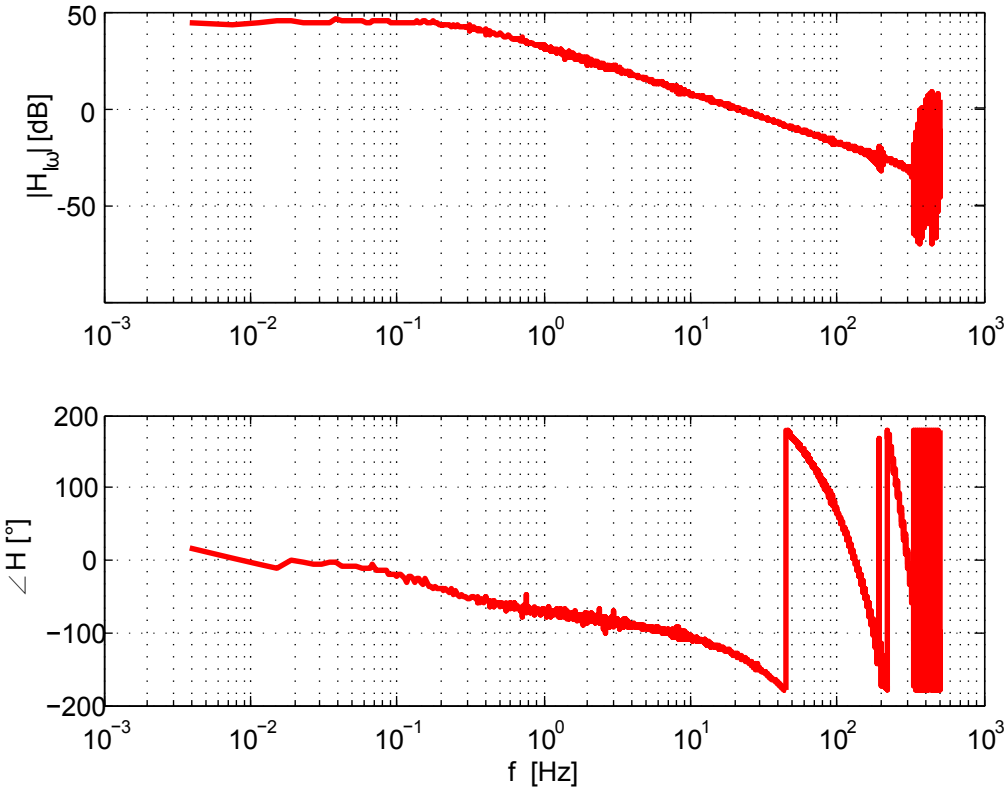


Figure 49: Output of the system excited by sine sweep

Estimated gains and phase shifts are presented in the form of the bode plot (figure 49). The magnitude plot —H— looks like a magnitude plot of the first order system. However the phase plot is influenced by transport delay. It causes rapid phase drop.

Output of `tfestimate` is used for the creation of `frd` (frequency-response) object, which is forwarded to the System Identification Tool. The tool is started by the command `Ident` (figure 50). It is necessary to cut out frequencies higher than 300 Hz to get rid of noise in the output signal (axis velocity). Data used for identification is in figure 51.

According to the magnitude response of the system, data is fitted to the transfer function in the form $H(s) = e^{-cs} \cdot \frac{b}{s+a}$; $(a, b, c) \in \mathbb{R}$. Several estimations have been made as seen in figure 52. Amplitude of the best transfer function fits measured data with 81,36% accuracy.

The estimated transfer function is in equation 27.

$$H(s) = e^{-0.001 \cdot s} \cdot \frac{438}{s + 1.75} \tag{27}$$

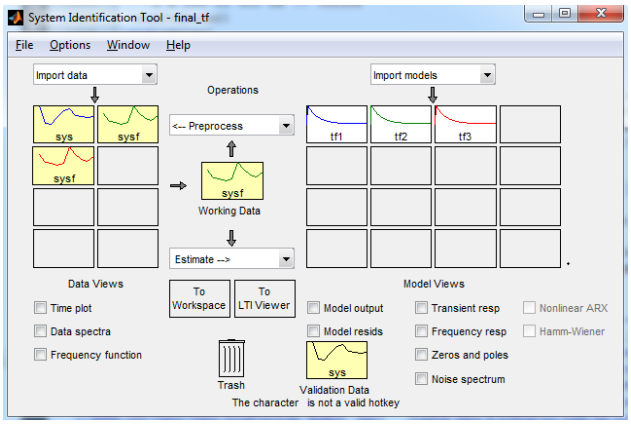


Figure 50: System Identification Tool

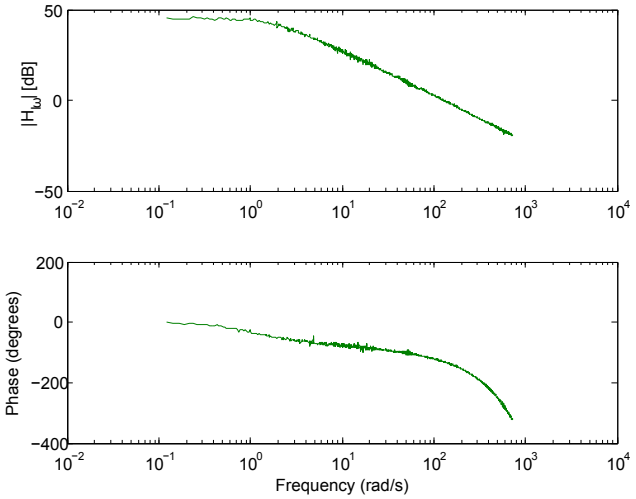


Figure 51: Data used for identification

The transfer function $H(s)$ is compared with the output of `tfestimate` function. The bode plot is in figure 53. It can be seen that phases do not match in higher frequencies. It means that delay 1 ms is not estimated correctly. As described in chapter 6.1, the delay of the position command is 8 ms. 2 ms takes cubic interpolation. Only brief information about inner functions of the controller were provided, but let us assume that the delay caused by interpolation in the current loop is much shorter than the interpolation delay in the position loop. Therefore the delay should be 2 ms shorter, which means 6 ms. The resulting transfer function is described by equation 28.

$$H(s) = e^{-0.006 \cdot s} \cdot \frac{438}{s + 1.75} \tag{28}$$

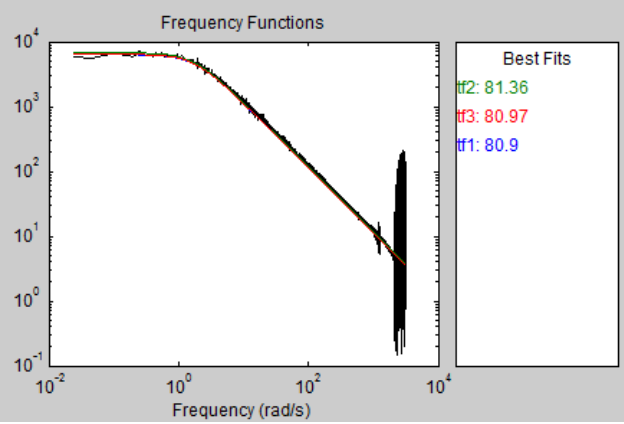


Figure 52: Estimated transfer functions

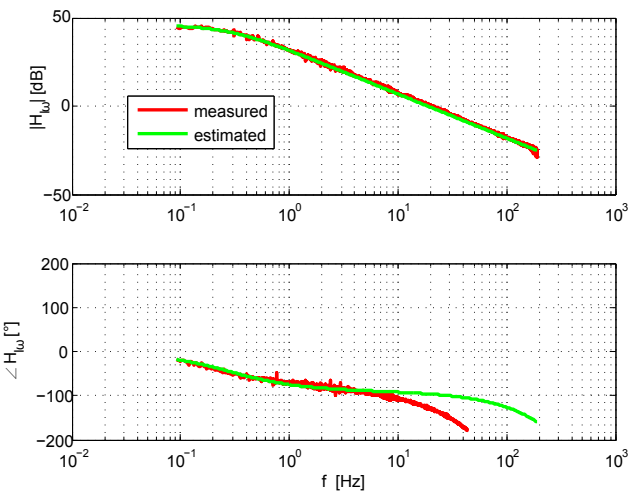


Figure 53: Bode plot of measured data and transfer function 27 with 1 ms delay

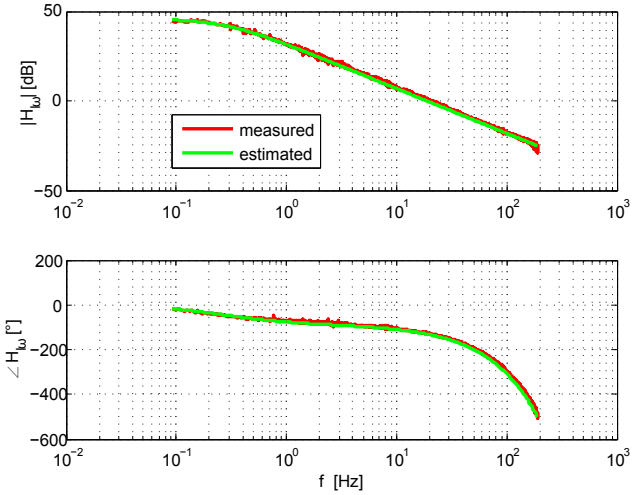


Figure 54: Bode plot of measured data and transfer function 28 with 6 ms delay

Comparison of transfer function 28 with measured data is in figure 54. Amplitudes and phases match well. Estimated transfer function 28 fits measured data adequately. The 6 ms transport delay is caused by the real time PC and during transfer of the command through EtherCAT to drives. The delay can be neglected, because regulation process takes place inside the controller. The final transfer function used for tuning of velocity controller gains is listed in the equation 29. Figure 55 shows the unit step response of the transfer function. The transfer function $H(s)$ contains physical representation of the Z axis and both motors including current controller.

$$H(s) = \frac{438}{s + 1.75} \tag{29}$$

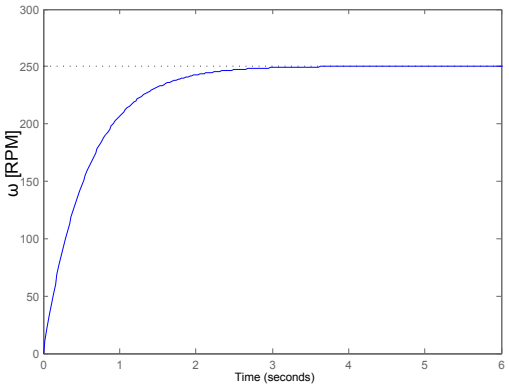


Figure 55: Unit step response of $H(s)$

9.2 Tuning of the velocity loop

Even if the transfer function of the controlled system is known, it might be difficult to balance gains of the PI controller to obtain the desired transient response. In the following chapter it is described how to substitute tuning of controller gains by two parameters. If the proportional gain k_p is changed, than integral gain k_i has to be changed too.

According to (Åström [2]), the better approach is to substitute both gains by parameters describing the undamped natural frequency of the system ω_0 and the relative damping ζ. The parameter ω_0 affects response speed and ζ changes the shape of the response. In the following lines it is described how to express the relation between k_p, k_i and ω_0, ζ.

The transfer function $H(s)$ obtained in the previous chapter is connected to the closed loop together with the PI controller $C(s)$. $H(s)$ is described by general equation 30.

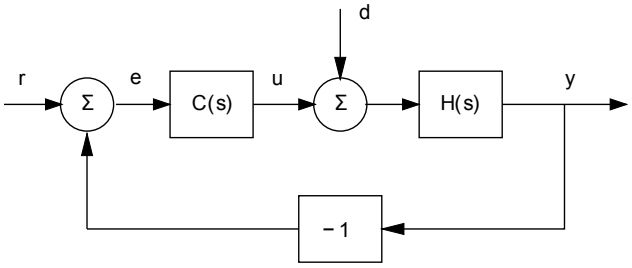


Figure 56: Closed loop with the velocity controller

$$H(s) = \frac{b}{s+a} \tag{30}$$

Parameters a and b are coefficients of the transfer function and s is the Laplace operator. The PI controller has the form as seen in the equation 31. The controller is considered as a continuous controller in the parallel form. It is obvious that controller is discrete, but the sampling frequency and additional details have not been provided. The sampling frequency is probably very high and therefore it is possible to take the controller as a continuous one.

$$C(s) = k_p + \frac{k_i}{s} \tag{31}$$

The closed loop is depicted in figure 56. The closed loop is described by the transfer function $L(s)$ from r to y in the equation 32.

$$L(s) = \frac{Y(s)}{R(s)} = \frac{H(s)C(s)}{1 + H(s)C(s)} = \frac{k_p b s + k_i b}{s(s+a)} = \frac{b(k_p s + k_i}{s^2 + (a + bk_p)s + bk_i)} \tag{32}$$

The characteristic polynomial of $L(s)$ is compared with the normalized second order polynomial (equation 33), where ζ denotes relative damping and ω_0 is the undamped natural frequency. Values of $\zeta < 1$ stands for underdamped system, $\zeta = 1$ is critically damped system and $\zeta > 1$ is overdamped system.

$$s^2 + (a + bk_p)s + bk_i = s^2 + 2\zeta\omega_0 s + \omega_0^2 \tag{33}$$

Parameters ω_0 and ζ are recalculated to the PI controller gains by equations 34 and 35.

$$k_p = \frac{2\zeta\omega_0 - a}{b} \tag{34}$$

$$k_i = \frac{\omega_0^2}{b} \tag{35}$$

Tuning of controller gains with parameters ω_0 and ζ provides good control over the response speed and the overshoot. As a starting point are chosen gains $k_p = 0{,}04$ and $k_i = 0{,}2$. Gains were obtained by manual tuning. Corresponding values are $\omega_0 = 9{,}3595$ and $\zeta = 1{,}0268$.

Tuning of controller gains is done in two steps. At first ω_0 will be changed to get desired rise time. Afterwards the overshoot will be compensated by changing ζ. The goal is to achieve rise time as short as possible with overshoot less than 10%. It is known that control

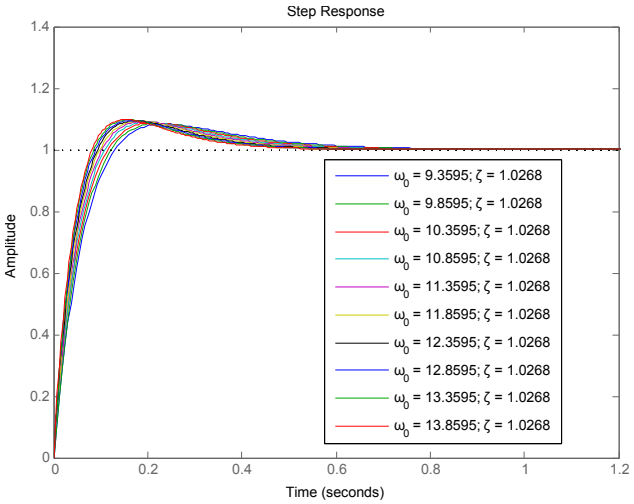


Figure 57: Step response with constant ζ

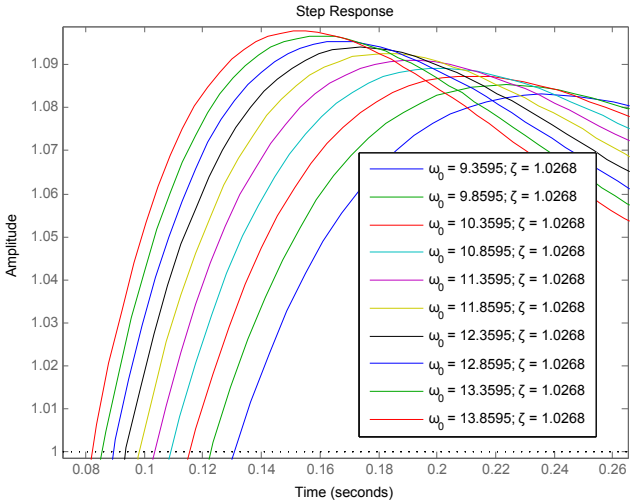


Figure 58: Detail of the step response

	1	2	3	4	5	6	7	8	9	10
ω_0	9.3595	9.859	10.36	10.86	11.36	11.86	12.36	12.86	13.36	13.86
ζ	1.0268	1.0268	1.0268	1.0268	1.0268	1.0268	1.0268	1.0268	1.0268	1.0268
k_p	0.039	0.042	0.044	0.046	0.049	0.051	0.053	0.056	0.058	0.060
k_i	0.2	0.221	0.245	0.269	0.294	0.321	0.348	0.377	0.4075	0.438

Table 9: Velocity controller tuning with constant ζ

is too aggressive and motors become noisy if $k_p > 0{,}05$. The only restriction is to keep k_p under that value.

Figure 57 shows effects of increasing ω_0 by 0.5 steps while $\zeta = 1{,}0268$. A detailed view is in figure 58. The rise time is decreasing, but overshoot is rising slightly. Corresponding values of controller gains are listed in table 9. Undamped frequency $\omega_0 = 10{,}35$ is chosen, because higher ω_0 has k_p values higher than 0,05.

Afterwards, ζ is changed by steps of 0,05 from 1 to 1,35 while $\omega_0 = 10{,}35$. Plots of the step response are in figure 60. Values of ω, ζ, k_p and k_i are in table 10. Value $\zeta = 1{,}1$ is

	1	2	3	4	5	6	7	8
ω_0	10.35	10.35	10.35	10.35	10.35	10.35	10.35	10.35
ζ	1.0	1.05	1.1	1.15	1.2	1.25	1.3	1.35
k_p	0.04326	0.04563	0.04799	0.05035	0.05272	0.05508	0.05744	0.05981
k_i	0.2446	0.2446	0.2446	0.2446	0.2446	0.2446	0.2446	0.2446

Table 10: Velocity controller tuning with constant ω_0

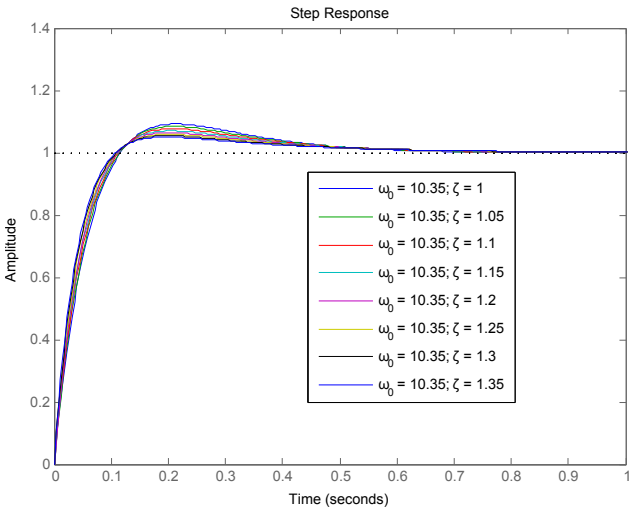


Figure 59: Step response with constant ω_0

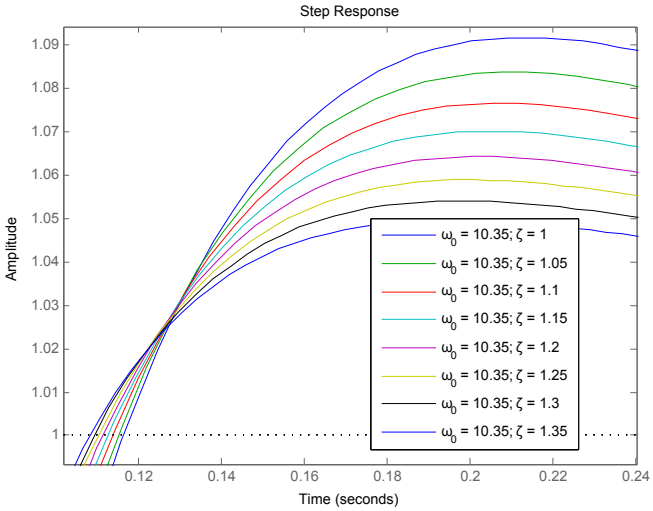


Figure 60: Detail of the step response

chosen with respect of $k_p < 0{,}05$. Result of tuning after recalculation is $k_p = 48000$ and $k_i = 24{,}5$. The final response is depicted in figure 61 and it has following properties:

Rise time:	0.0781 s
Settling time:	0.4848 s
Overshoot:	7.6602 %

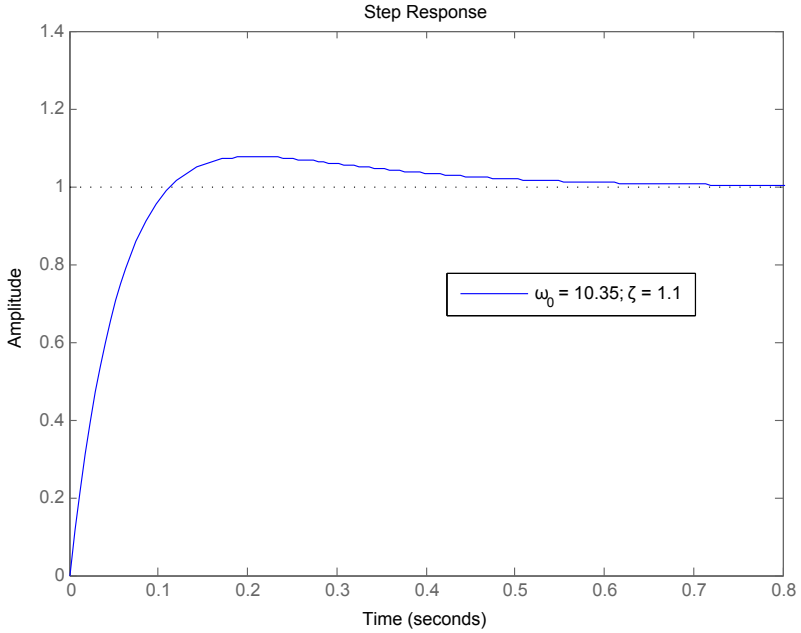


Figure 61: Step response of the system with tuned velocity controller

s_d [mm]	j [ms^{-3}]	a_{max} [ms^{-2}]	v_{max} [ms^{-1}]	t_{tot} [ms]
6,19	1,04166 $\cdot 10^3$	6,25	0,179	133

Table 11: The punch parameters

9.3 Tuning of the position loop

The tuning of the position loop will be done with respect to the reference trajectory. The main target is to track the reference trajectory as well as possible. The response to disturbance represented by the impact to the sheet of metal is good even if control loops are tuned poorly. Therefore it is not necessary to test it. The reference trajectory is in figure 62. Parameters characterising the slope are in table 11.

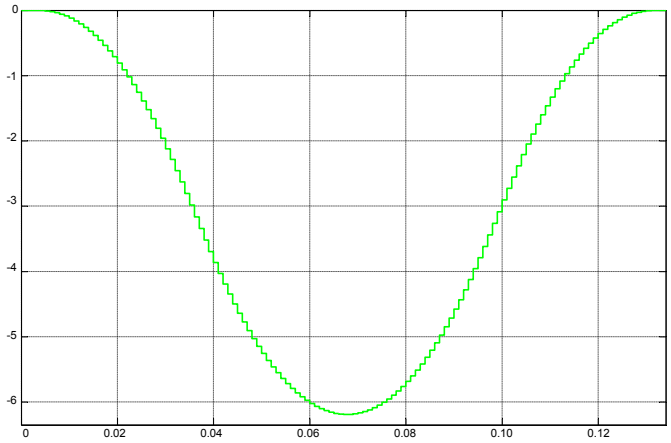


Figure 62: The slope used for tuning

Tuning of the position loop is implemented in the Simulink model. The model is depicted in figure 63. The transfer function $H(s) = \frac{438}{s+1.75}$ of the tuned system is used as a description. It covers the current controller, both motors and Z axis. The velocity controller in the block *PI speed* is the same as the one in the chapter 8.4. It has gains obtained by tuning in the previous chapter. The controller can be seen in figure 64. The only difference is presence of *Interpolation* block which converts discrete output of the position controller to the continuous signal.

The position controller in the block *Real time PC* is implemented as described in chapter 7. The source code of the controller is in the code listing 10. It is executed every 1 ms. Discrete

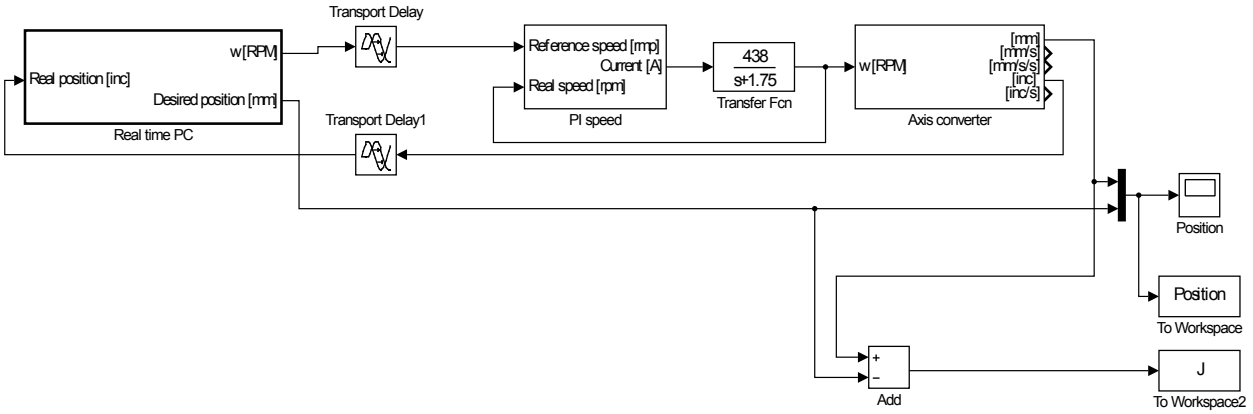


Figure 63: Simulink model for position tuning

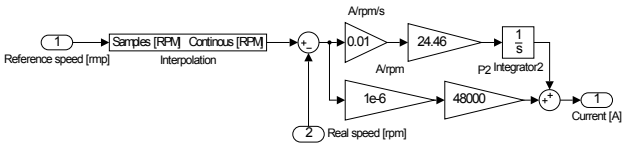


Figure 64: Velocity PI controller

sampling time in the Simulink is implemented by usage of Atomic subsystem with sample time 1 ms. The controller is depicted in figure 65. The block called *Axis converter* serves

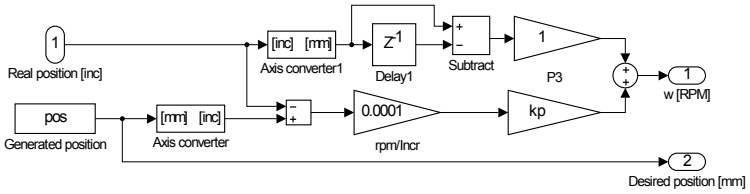


Figure 65: Position controller inside the real time PC

only for conversion between units. Tuning itself is conducted as an optimization task. At first the simulation is executed with initial setting of the proportional gain k_p. The initial value is set to $k_p = 800$. It is the value obtained by manual tuning.

Then the Matlab script in the code listing E.1 calls the optimization function *fminsearch*. The optimization function executes perpetually the function *Criterion.m* listed in the code listing E.2. The script *Optimization.m* starts the simulation. Output of the simulation is a vector $\overrightarrow{J} = (J_1, J_2, \cdots, J_n)$. The vector $\overrightarrow{J}$ contains samples of differences between the real and reference trajectory.

The $\overrightarrow{J}$ is processed by the function f in the equation 36. The result of f is passed back to the *fminsearch*. The *fminsearch* decides how to change the k_p and executes *Criterion.m* perpentualy, until the minimal value of f is found.

$$f = \sum_{i=1}^{N} J_i^2 \tag{36}$$

The result of the optimization is in figure 66. The proportional gain k_p is gradually increased until instability occurs. Than k_p is decreased to get rid of the unstable behaviour. The final value is $k_p = 3480$. It is quite an aggressive set-up, oscillations in the steady state phase are still present.

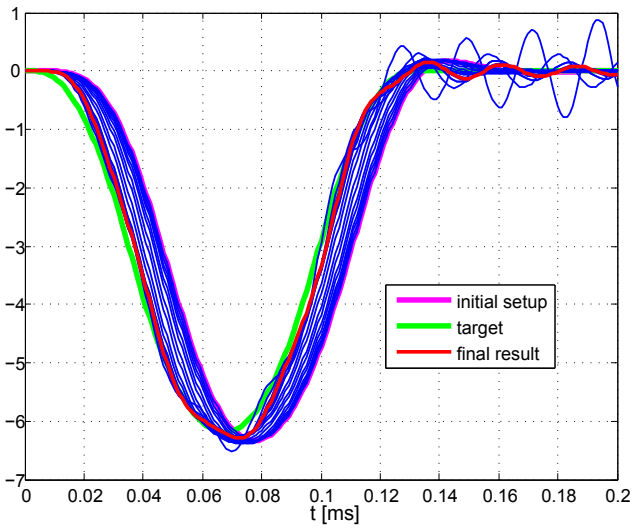


Figure 66: Optimization process

Therefore the equation 36 is changed to the weighting function listed in the equation 37, where k is the first sample of the steady state phase. Oscillations at the end of the trajectory have 100 times bigger influence on the value f than the rest of the samples. The result of optimization is in figure 67. The weighting decreased oscillations and the final value of the proportional gain is $k_p = 2800$. The overshoot at the end of the movement is 0,06 mm, which is a negligible value.

$$f = \begin{cases} \sum_{i=1}^{N} J_i^2 & \text{if } i < k \\ 100 \cdot \sum_{i=1}^{N} J_i^2 & \text{if } i \geq k \end{cases} \tag{37}$$

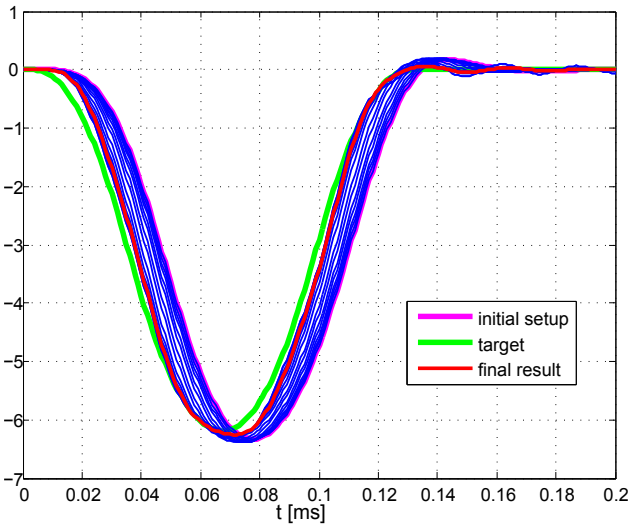


Figure 67: Optimization process with the weighting function

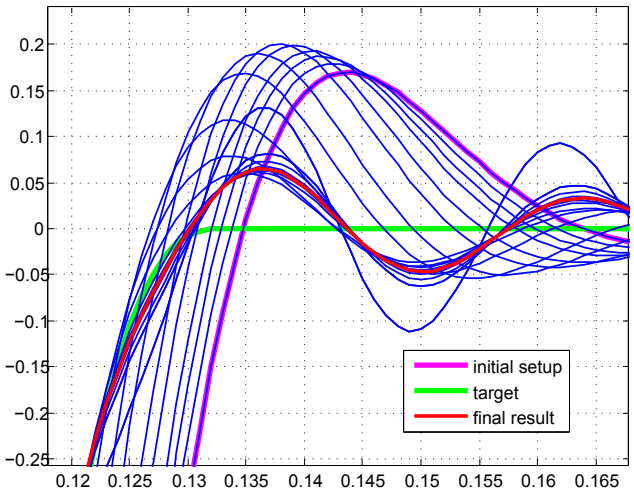


Figure 68: Detail view of the optimization process with the weighting function

However the k_p does not correspond with the value obtained by the manual tuning. It might be caused by insufficient knowledge of the inner properties of drives. The connection between the real time PC and drives might be modelled incorrectly.

9.4 Summary

The transfer function of Dynapunch is obtained by analysing the frequency response. The Dynapunch is identified as a first order system with the transport delay. The transfer function is used for tuning velocity and position controllers.

Controllers inside drives are considered as continuous, although they are discrete. It is assumed that the sampling frequency of controllers is high enough for considering controllers as continuous. Proportional and integral gain values of the velocity controller correspond with the values obtained by the manual tuning. It proves that usage of continuous controllers in the tuning process does not affect the result of tuning.

However, the value of proportional gain $k_p = 2800$ does not match with the value $k_p = 800$ which is obtained by the manual tuning. It is not possible to analyse the issue in more detail, because there is a lack of information from the side of the manufacturer of drives. Implementation of position controller outside the drive requires sending the velocity command to drives through the *Velocity offset* input (figure 28). There is a low pass filter on the way of the command. The filter is turned off according to the documentation, but it is possible

that some unknown factors can affect the value of the velocity command. This conclusion is deduced from the fact, that commissioning of the *Velocity offset* input is a matter of a lot of changes in the drives firmware during placement of the position controller outside drives. Proper implementation of the position controller outside drives will probably require additional communication with the drive manufacturer.

Results of tuning have not been tested on the real system, because there are time schedule related issues described in the following chapter. Even if testing is not done, controllers are successfully tuned and tested during manual tuning.

10 Remark

Control loop gains parameters obtained in the chapter 9 are not tested on the real system and compared with values obtained by the manual tuning. There was not enough time to do it. The working schedule was continuously delayed by fixing bugs in the drives firmware. Although the functionalities of drives were listed in the datasheet, they had not been implemented properly and drives fell into an error state very often.

The drives were originally developed for driving industrial robots. That kind of application did not push drives to the borders of their operational limits. However, application in a punch press machine has different demands. Drives are required to switch rapidly from minimal to maximum power and have to cope with large position errors during the impact of the punch. It frequently results in the activation of safety measures in drives. Measures provide protection against damage of servo drives or prevents the robot from movement out of the defined trajectory.

These safety measures are not applicable to punch press machines. It is very common to have sudden changes of required torque followed by a large position error during the impact into the sheet of metal. Therefore it was necessary to change safety measures to a less restrictive form. The only way how to do it was to report problems to the manufacturer of drives. The manufacturer had to send new versions of the drive's firmware which usually took more than one week. This is the reason why the testing on the real system was not finished completely. It is definitely not a mistake of the manufacturer. It is normal that problems might appear if new technology is implemented for the first time.

11 Conclusion

The work done during the thesis results in the functional implementation of servo drives on the prototype of the punch press machine. Dynapunch is brought to the state where it is able to make coordinated movements with both axes and perform testing patterns.

Several tasks are solved. The trajectory generator is rewritten in a more usable form. The algorithm for the motion anticipation has been created. Completion of both tasks allowed the synchronization of the Z and X axes together. Axis synchronization allows the start of movements with one axis while the other is finishing movement. This leads to improvement of the punching speed.

The second servomotor is connected to the Z axis and solution for motors synchronization is implemented. The most significant issue is that servo drives are not supporting any communication between drives. The communication problem is solved by moving position control loop outside drives into the real time PC. This solution proves to be effective, but it also exposes issues which appeared in the next stage of work.

Drives are tuned manually. Manual tuning gives sufficient results, but it requires a large amount of time. Therefore the mathematical model of the machine is created in the Simulink environment. However, the model does not approximate the real system well, because of the lack of information about the inner properties of servo drives.

The method for system identification is applied instead of creation of the model. The transfer function is obtained and used for the tuning of velocity and position controller. The target of tuning is to have the fastest transient response with minimal overshoot. The identification method is successful, because values of proportional and integral gains of the velocity controller are very similar to the values obtained by manual tuning. It is not necessary to tune drives for good reaction to the disturbance, because the punch press is able to perforate steel sheets up to 6 mm without problems.

The tuning of the position control loop is done on the Simulink model. The position controller in the real time PC is implemented as well as velocity controller inside drives. The tuning is focused on the tracking of the reference trajectory. The appropriate value of the position proportional gain is found as a solution of the optimization task. The trajectory obtained from the simulation fits the trajectory obtained by the manual tuning. However

values of proportional gains do not match. The difference of gains is major and could not be caused by implementation of the model or by an error during system identification. As the main reason is considered an undocumented element in the path of the velocity command. Therefore it is recommend to use values obtained by the manual tuning from chapters 8.1 and 8.2. Values obtained by the manual tuning are sufficient for the control of axes.

Further work on Dynapunch should be focused on the examination of the difference between values obtained by the manual tuning and system identification method. Especially signal path of the *Velocity offset* command should be examined carefully. If the difference between methods is eliminated, the system identification method will became more reliable and precise than the manual tuning method.

It is not recommended to improve the model described in the chapter 8.4, because creation of the model which is approximating reality well proved to be very difficult. Better results gives the method of system identification.

The next step in the development of Dynapunch is to create an adaptive punching algorithm which will lead to improved productivity and energy savings. The maximum speed of the Z axis is 500 strokes per minute or 600 strokes per minute if reduced stroke length is used. This result provides good base for further development. Servo drives proved to be capable of powering the prototype of the punch press machine.

Literature

[1] SERRUYS, Wim; *Blecharbeitung, Stand der Technik*; LVD Company n. v., Nederlands, Belgium 2006. 111 pages, ISBN: 9789080722491

[2] ÅSTRÖM, Karl; *Control System Design*, [online], Santa Barbara, California 2002. 333 pages, [cit. 2014-10-15]. Available from: `http://neutron.ing.ucv.ve/eiefile/Control%20I/Astrom_notas.pdf`

[3] ETHERCAT TECHNOLOGY GROUP; *EtherCAT - the Ethernet Fieldbus*, [online], Nuremberg, Germany 2014. [cit. 2014-10-15]. Available from: `http://www.ethercat.org/en/technology.html`

[4] CONTROLENG CORPORATION; *SERVOsoft Help, Motion Profile*, [online]. Maple, Ontario 2011. ISBN: 978-1-4577-0755-1. [cit. 2014-10-15]. Available from: `http://www.controleng.ca/servosoft/SSHelp1033/source/MotionProfile.htm`

[5] AMD; *AMD Geode^{TM} LX Processors Data Book*, [online]. Sunnyvale, California, May 2008. 678 pages, [cit. 2014-10-15] Available from: `http://www.versalogic.com/support/Downloads/PDF/LXManualMay08.pdf`

[6] RACCIU, MANTAGEZZA; *RTAI User Manual 3.4*, [online], October 2006. [cit. 2014-10-15] Available from: `https://www.rtai.org/?About_RTAI`

[7] MAVILOR; *AC Servo Motors BL 110/140/190 Series*, [online]. Barcelona,Spain 2001. [cit. 2014-10-15] Available from: `http://www.mavilor.es/pdf_products/bl100_series_sc.pdf`

[8] MAVILOR; *AC Servo Motors BL 40/50/70 Series*, [online]. Barcelona,Spain 2001. [cit. 2014-10-15] Available from: `http://www.mavilor.es/pdf_products/bl40_series_sc.pdf`

[9] IRT SA; *IRT Operating manual, Drives 2000 S-AT, 4000 S-AT*, [online], Neuchâtel 2013. [cit. 2014-10-15] Available from: `http://www.irtsa.com/IMG/pdf/op2_4_0913e.pdf`

[10] REIS GMBH & CO MASCHINENFABRIK OBERNBURG; *EtherCAT with Drive profile CiA402*, [datasheet], 39 pages, Obernburg 2012.

[11] BECKHOFF AUTOMATION GmbH; *EL1002, 2-channel digital input terminal 24 V DC*, [online], Verl, Germany 2010. [cit. 2014-10-15] `http://download.beckhoff.com/download/Document/Catalog/Main_Catalog/english/separate-pages/EtherCAT/EL1002.pdf`

Appendix A Attached CD

```
CD
|    Application of servo drives on the prototype of the punch press machine.pdf
|
├─────Drives tuning
|        Motor_test.slx
|        Simulation_scheme.slx
|        step_2000_3_curr_act.txt
|        step_2000_3_velo_act.txt
|        step_z_irt.m
|
├─────Plotting tool
|        config.ini
|        SCOPE_2025_3mm.SCP
|        scope_graph_importer.m
|        scope_graph_maker.m
|
└─────System_identification
     ├─────Position_loop_tuning
     |        Criterion.m
     |        Optimization.m
     |        posData.mat
     |        position.m
     |        position_controller_tune.slx
     |
     ├─────Sweeps
     |    |    bigdata.mat
     |    |    bodes_tfestimate.m
     |    |    bode_comparison.m
     |    |    load_scopes.m
     |    |    plots.m
     |    |    tfmodels.sid
     |    |
     |    └─────data
     |              1_sweep_01_1_20.txt
     |              1_sweep_01_1_40.txt
     |              1_sweep_09_5_40.txt
     |              sweep_14_20_200.txt
     |              sweep_199_400_1000.txt
     |              sweep_19_30_200.txt
     |              sweep_29_40_200.txt
     |              sweep_39_100_1000.txt
     |              sweep_4_10_40.txt
     |              sweep_4_10_80.txt
     |              sweep_99_200_100.txt
     |              sweep_99_200_1000.txt
     |              sweep_9_15_120.txt
     |
     └─────Velocity loop tuning
              tuning.m
```

Appendix B Axis synchronization tests

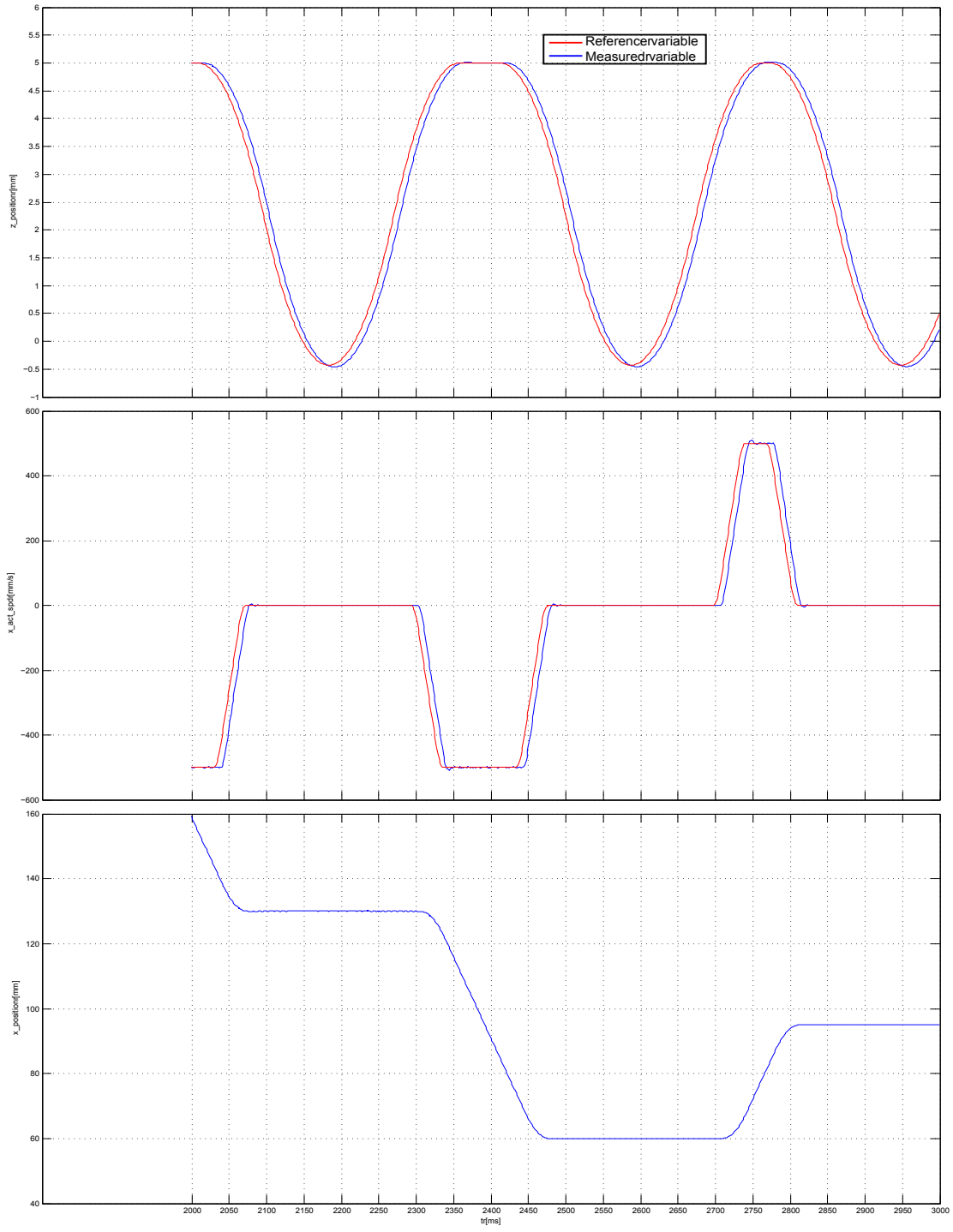


Figure 69: Axes synchronization test, Parameters of the Z axis: hover height $h = 5$ mm, die penetration $d_p = 1.2$ [mm], jerk $j = 1.041$ ms^{-3}, max. acceleration $a_{max} = 62.5$ ms^{-2}, max. speed $v_{max} = 0.179$ ms^{-1}. Parameters of the X axis: stroke distance $s_d = 70$ mm, jerk $j = 3500$ [ms^{-3}], max. acceleration $a_{max} = 35$ ms^{-2}, max. speed $v_{max} = 0.5$ ms^{-1}

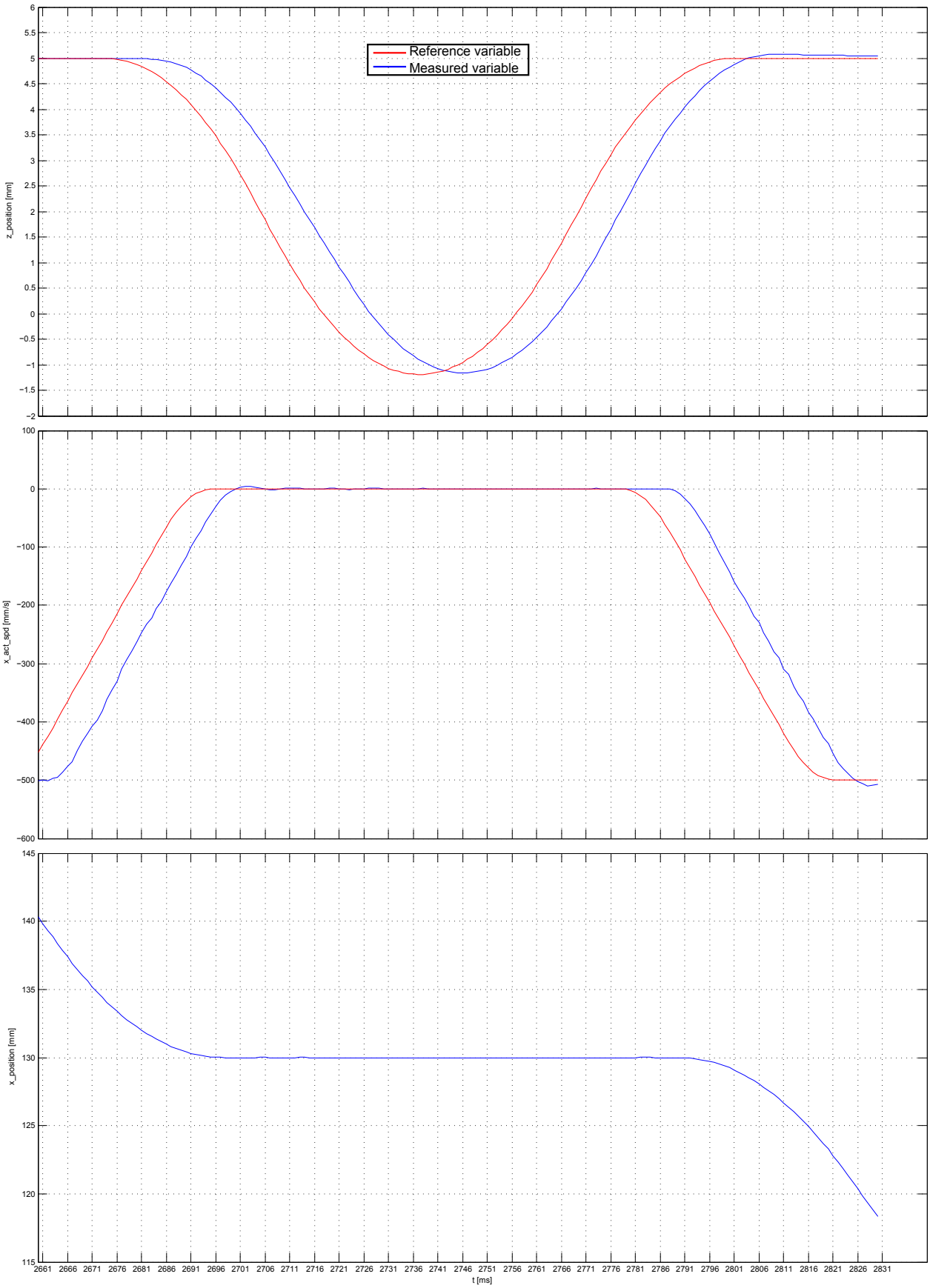


Figure 70: Axes synchronization test, Parameters of the Z axis: hover height $h = 5$ mm, die penetration $d_p = 1.2$ [mm], jerk $j = 1.041$ ms^{-3}, max. acceleration $a_{max} = 62.5$ ms^{-2}, max. speed $v_{max} = 0.179$ ms^{-1}. Parameters of the X axis: stroke distance $s_d = 70$ mm, jerk $j = 3500$ [ms^{-3}], max. acceleration $a_{max} = 35$ ms^{-2}, max. speed $v_{max} = 0.5$ ms^{-1}

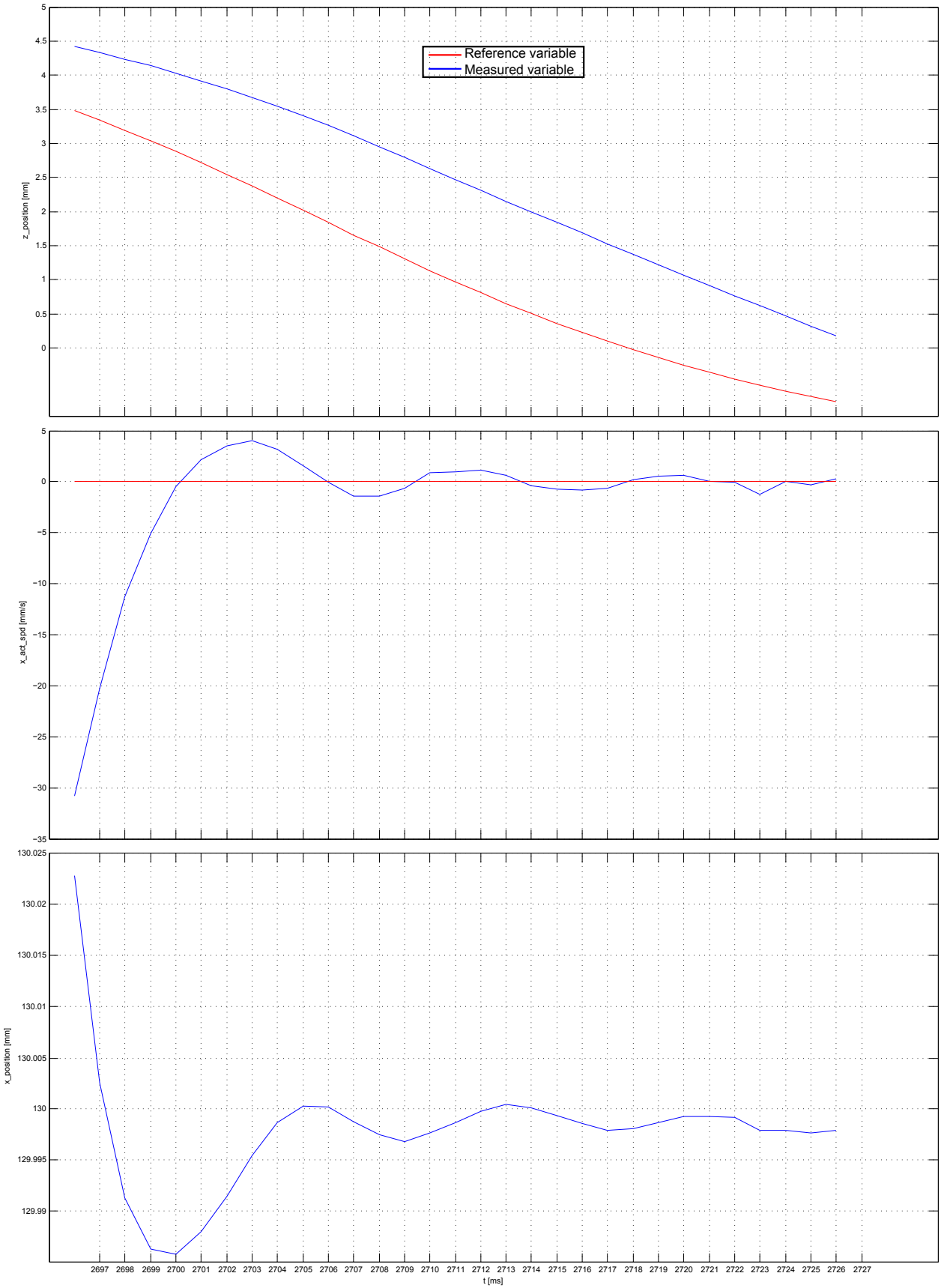


Figure 71: Axes synchronization test, Parameters of the Z axis: hover height $h = 5$ mm, die penetration $d_p = 1.2$ [mm], jerk $j = 1.041$ ms^{-3}, max. acceleration $a_{max} = 62.5$ ms^{-2}, max. speed $v_{max} = 0.179$ ms^{-1}. Parameters of the X axis: stroke distance $s_d = 70$ mm, jerk $j = 3500$ [ms^{-3}], max. acceleration $a_{max} = 35$ ms^{-2}, max. speed $v_{max} = 0.7$ ms^{-1}

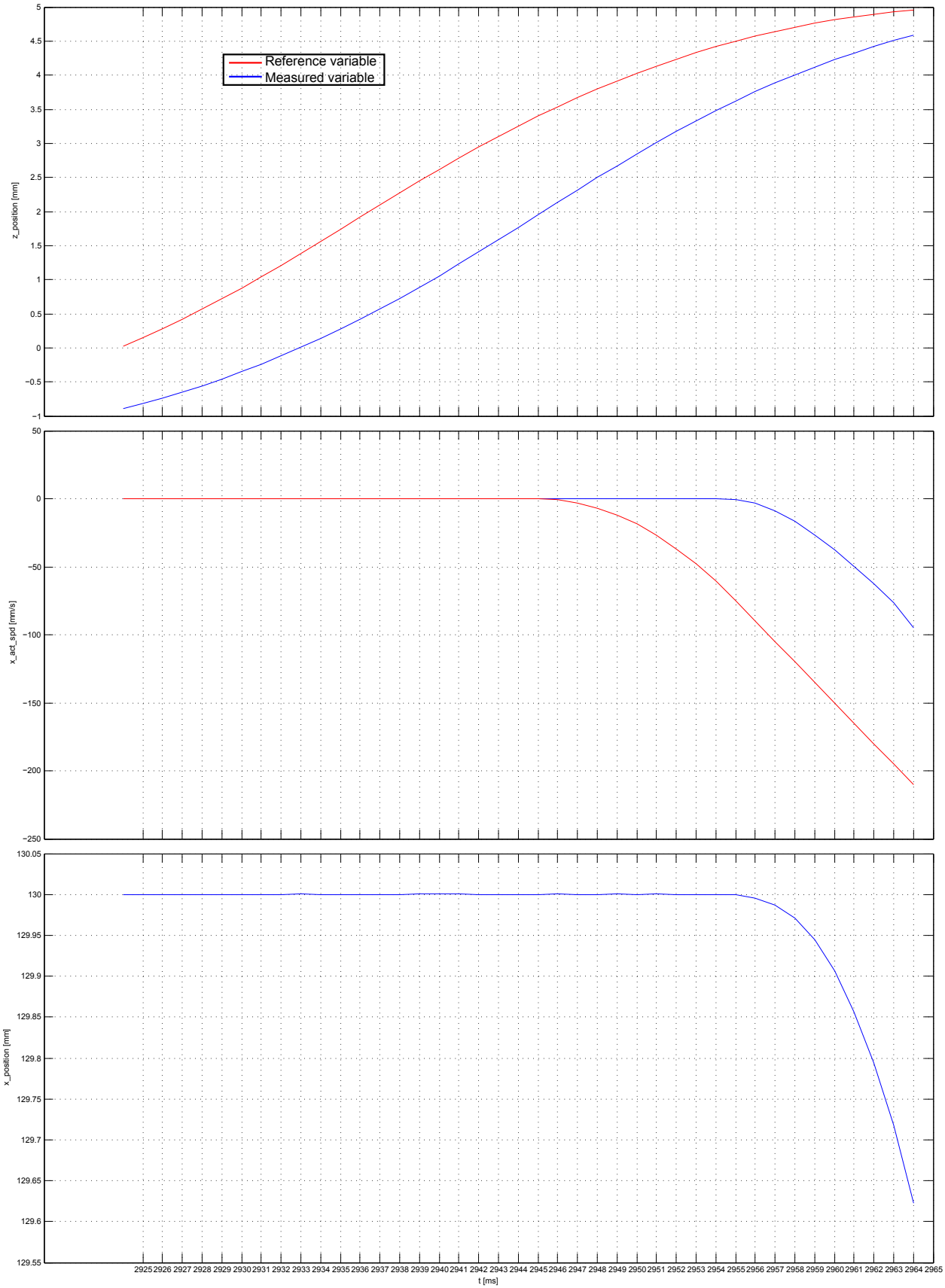


Figure 72: Axes synchronization test, Parameters of the Z axis: hover height $h = 5$ mm, die penetration $d_p = 1.2$ [mm], jerk $j = 1.041$ ms^{-3}, max. acceleration $a_{max} = 62.5$ ms^{-2}, max. speed $v_{max} = 0.179$ ms^{-1}. Parameters of the X axis: stroke distance $s_d = 70$ mm, jerk $j = 3500$ [ms^{-3}], max. acceleration $a_{max} = 35$ ms^{-2}, max. speed $v_{max} = 0.7$ ms^{-1}

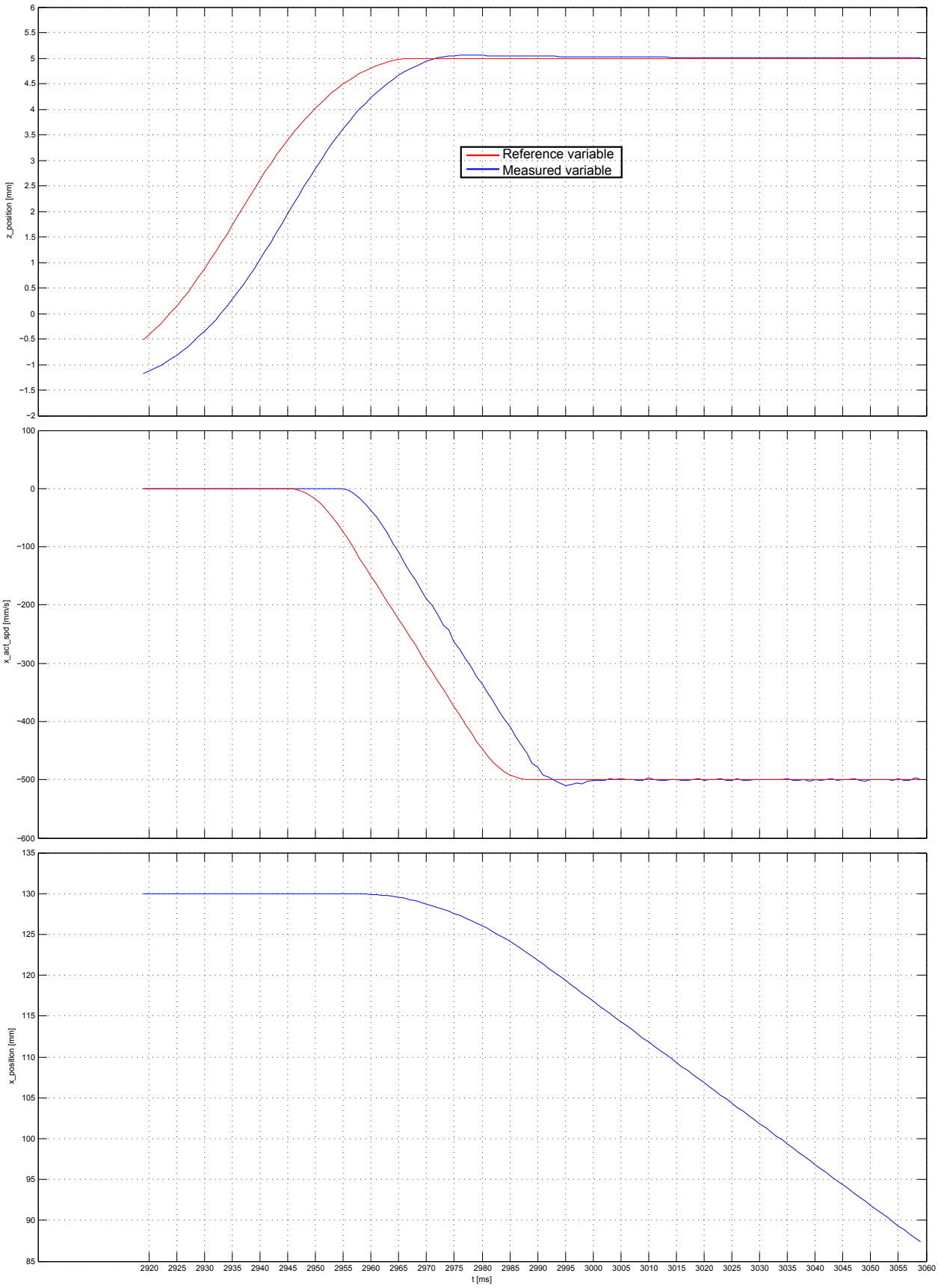


Figure 73: Axes synchronization test, Parameters of the Z axis: hover height $h = 2.5$ mm, die penetration $d_p = 1.2$ [mm], jerk $j = 1.041$ ms^{-3}, max. acceleration $a_{max} = 62.5$ ms^{-2}, max. speed $v_{max} = 0.179$ ms^{-1}. Parameters of the X axis: stroke distance $s_d = 100$ mm, jerk $j = 3500$ [ms^{-3}], max. acceleration $a_{max} = 35$ ms^{-2}, max. speed $v_{max} = 0.5$ ms^{-1}

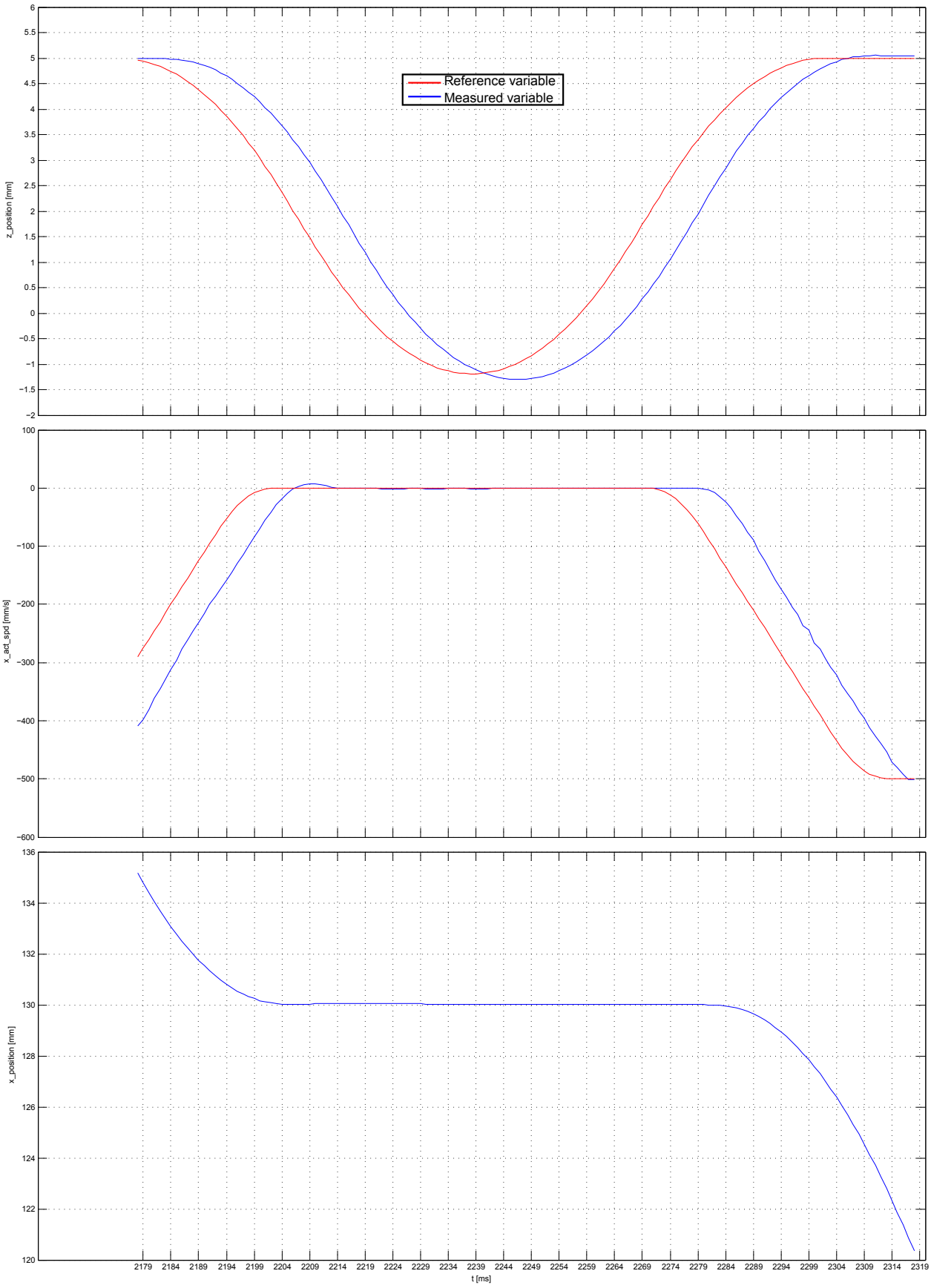


Figure 74: Axes synchronization test, Parameters of the Z axis: hover height $h = 5$ mm, die penetration $d_p = 1.2$ [mm], jerk $j = 1.041$ ms^{-3}, max. acceleration $a_{max} = 62.5$ ms^{-2}, max. speed $v_{max} = 0.7$ ms^{-1}. Parameters of the X axis: stroke distance $s_d = 10$ mm, jerk $j = 3500$ [ms^{-3}], max. acceleration $a_{max} = 15$ ms^{-2}, max. speed $v_{max} = 0.5$ ms^{-1}

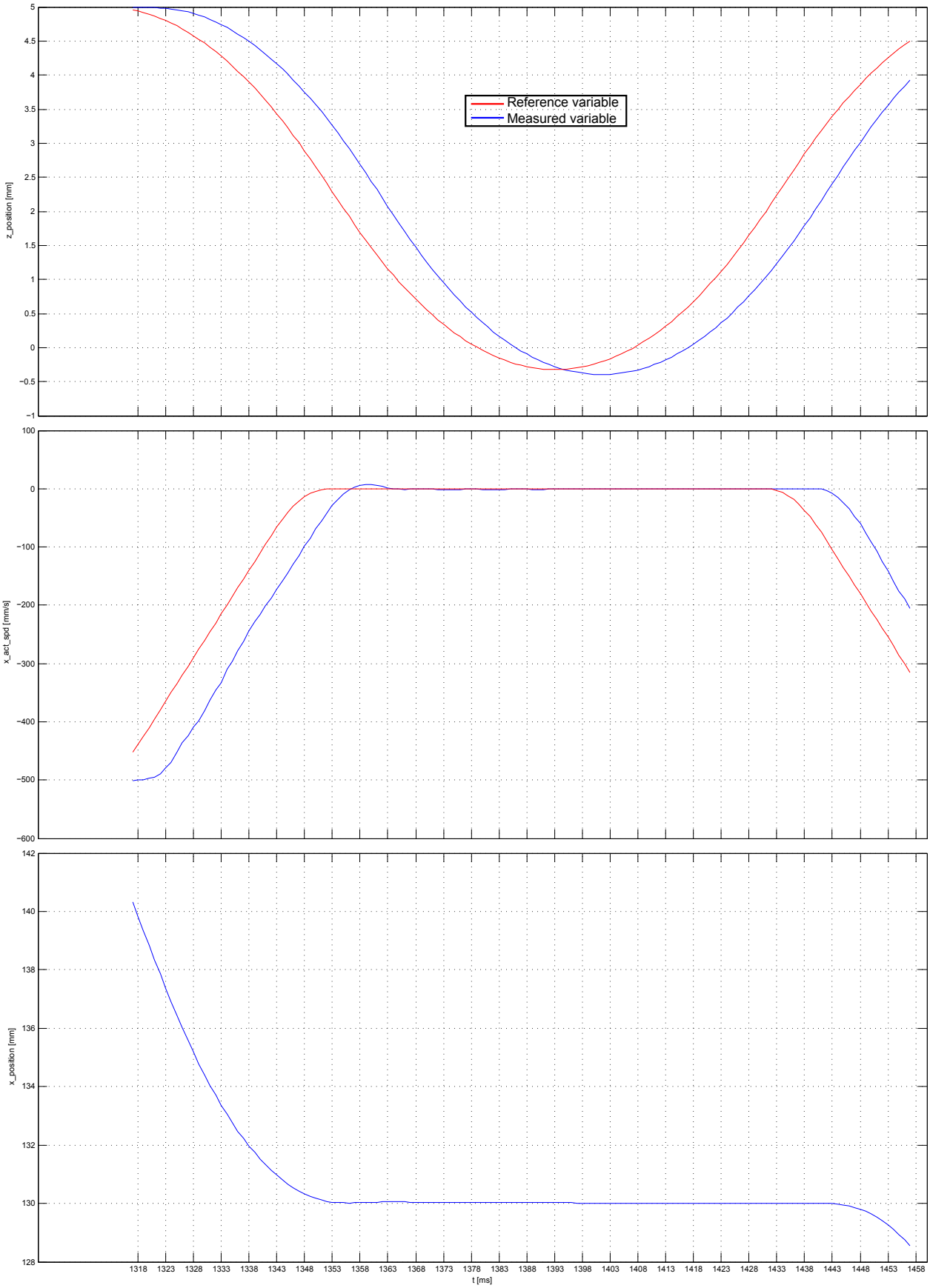


Figure 75: Axes synchronization test, Parameters of the Z axis: hover height $h = 2.5$ mm, die penetration $d_p = 0.3$ [mm], jerk $j = 0,5025$ ms^{-3}, max. acceleration $a_{max} = 32.5$ ms^{-2}, max. speed $v_{max} = 0.122$ ms^{-1}. Parameters of the X axis: stroke distance $s_d = 10$ mm, jerk $j = 3500$ [ms^{-3}], max. acceleration $a_{max} = 35$ ms^{-2}, max. speed $v_{max} = 0.7$ ms^{-1}

Appendix C Manual tuning plots

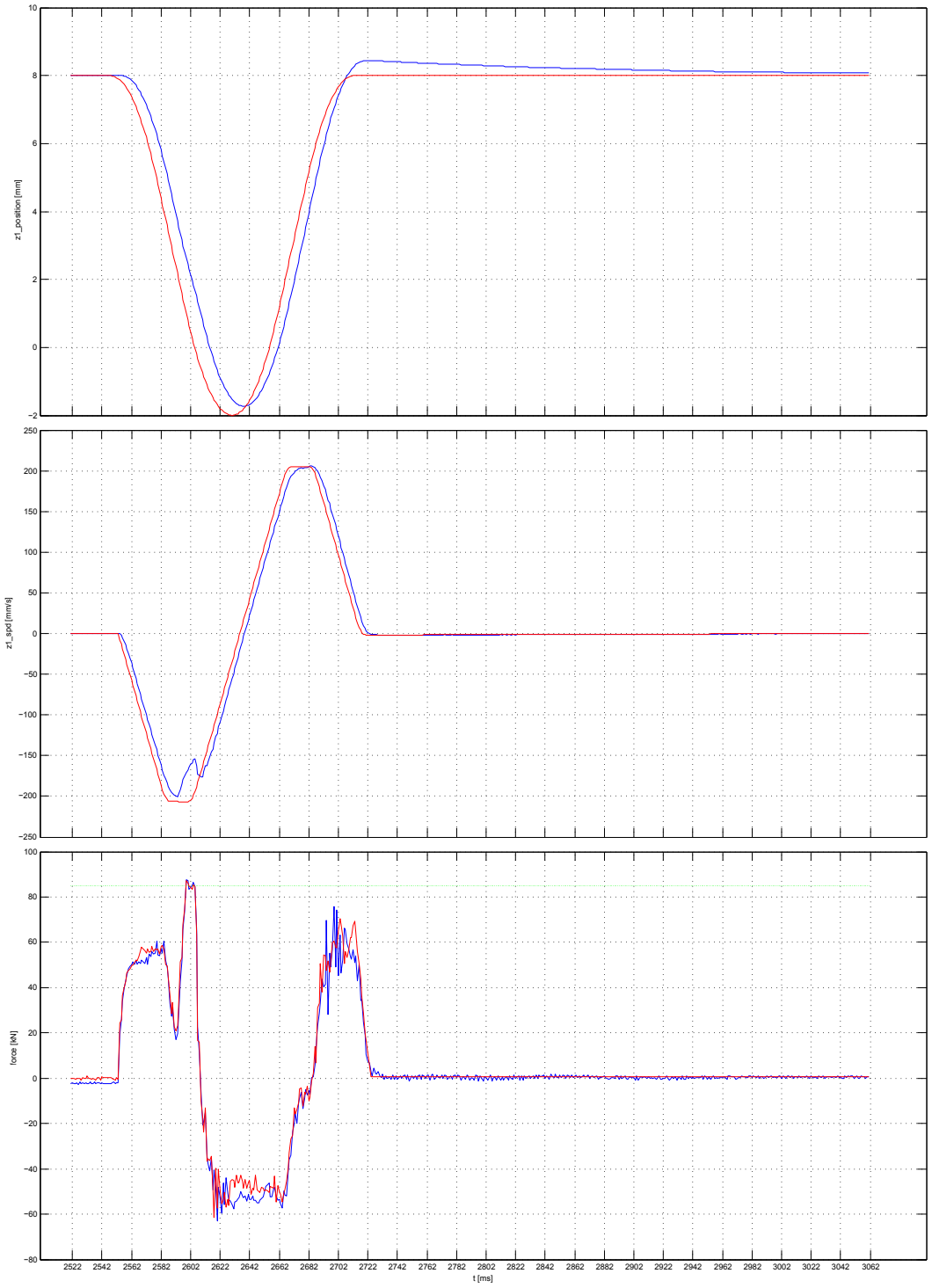


Figure 76: The motion profile of the Z axis without gain scheduling

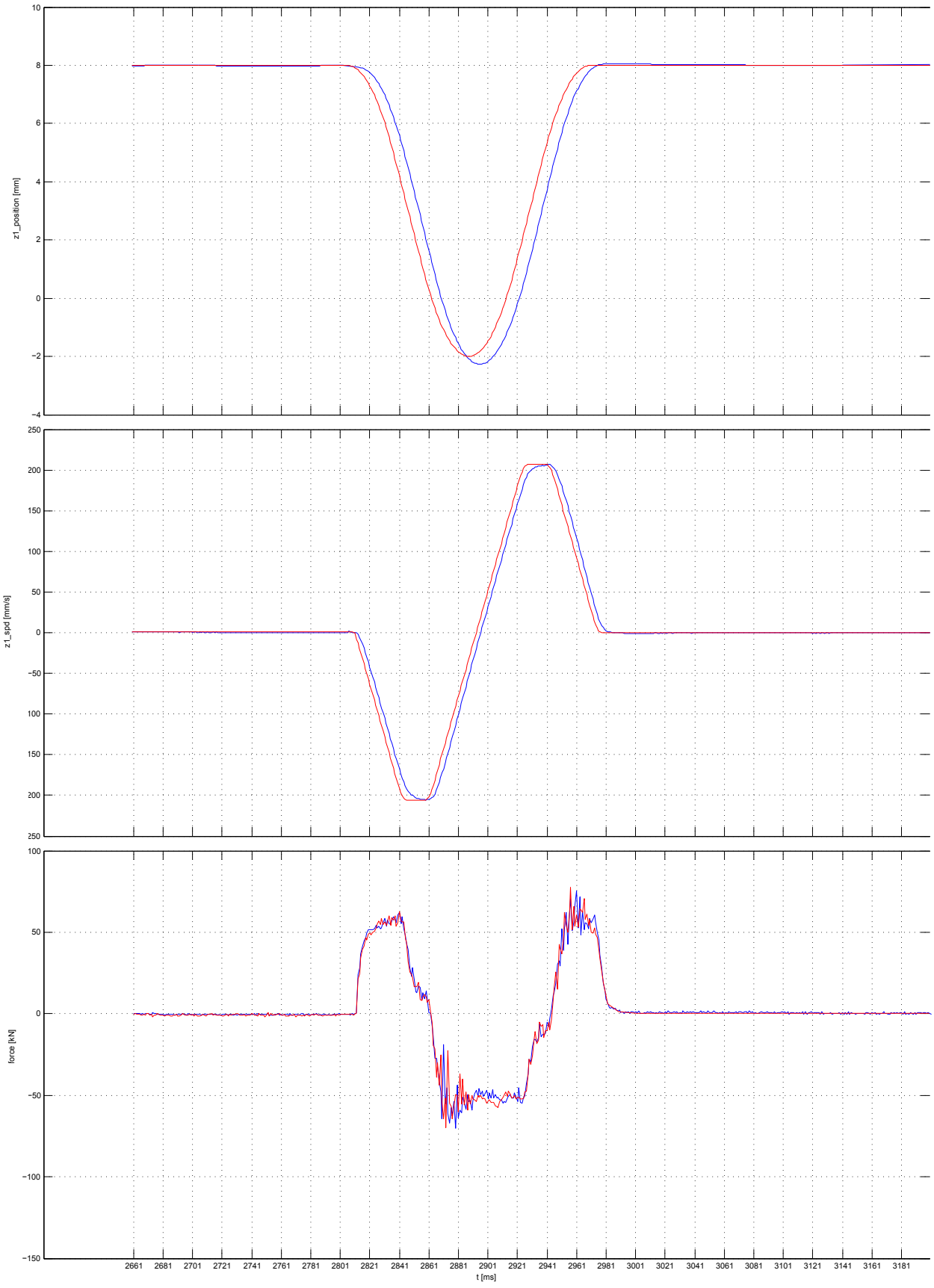


Figure 77: The motion profile of the Z axis with gain scheduling

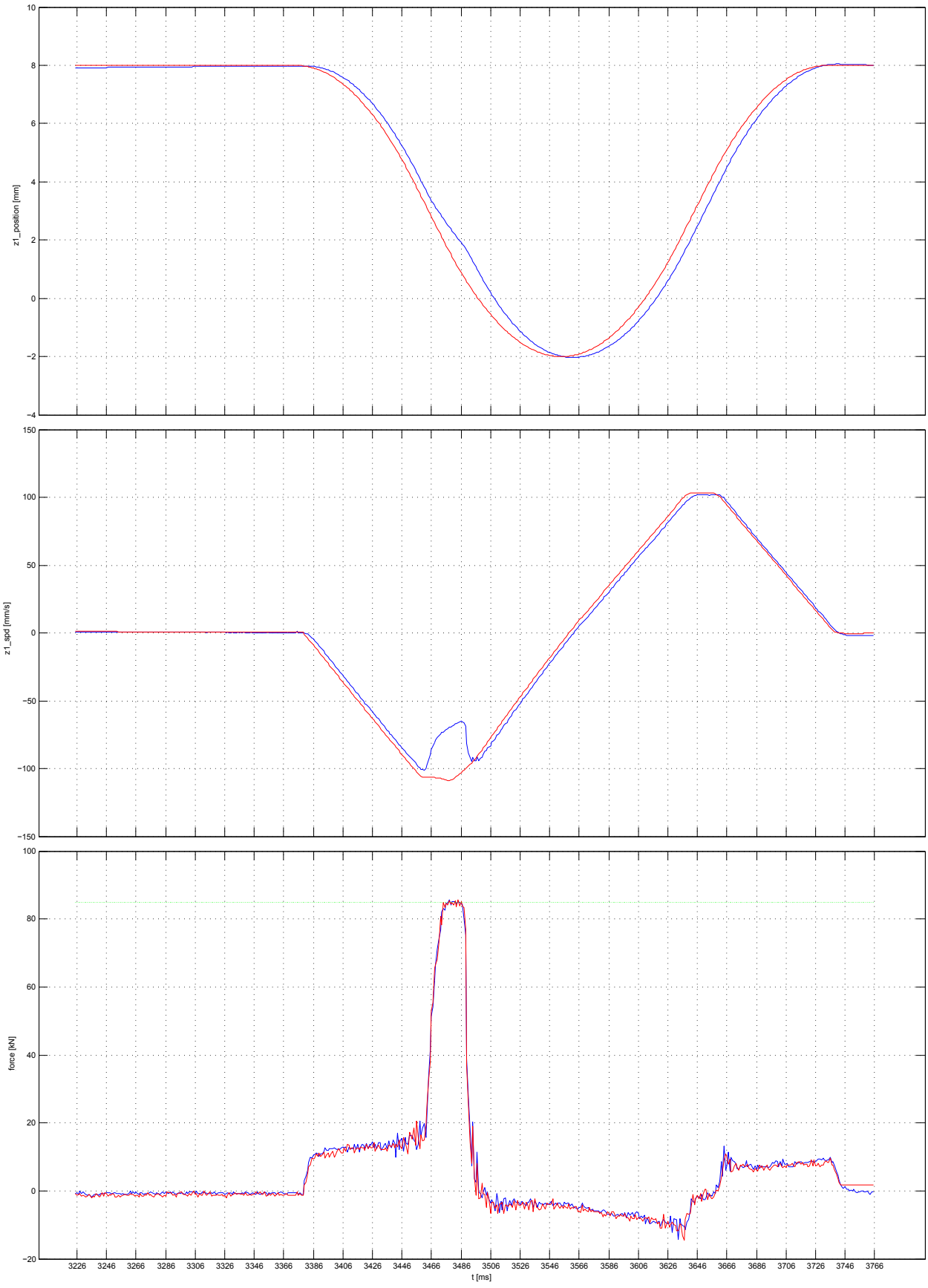


Figure 78: Slow punch through 5 mm steel plate

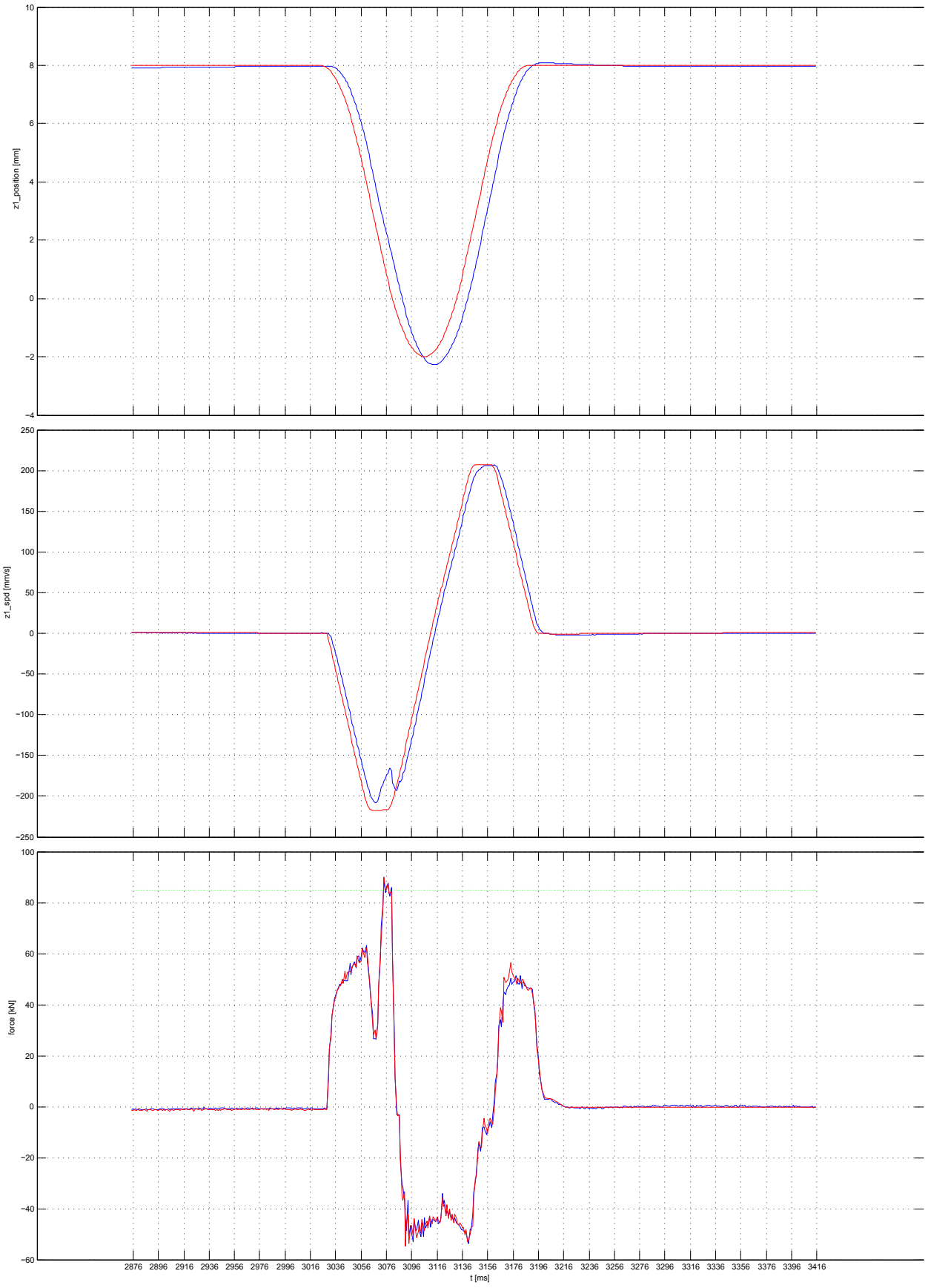


Figure 79: Fast punch through 5 mm steel plate

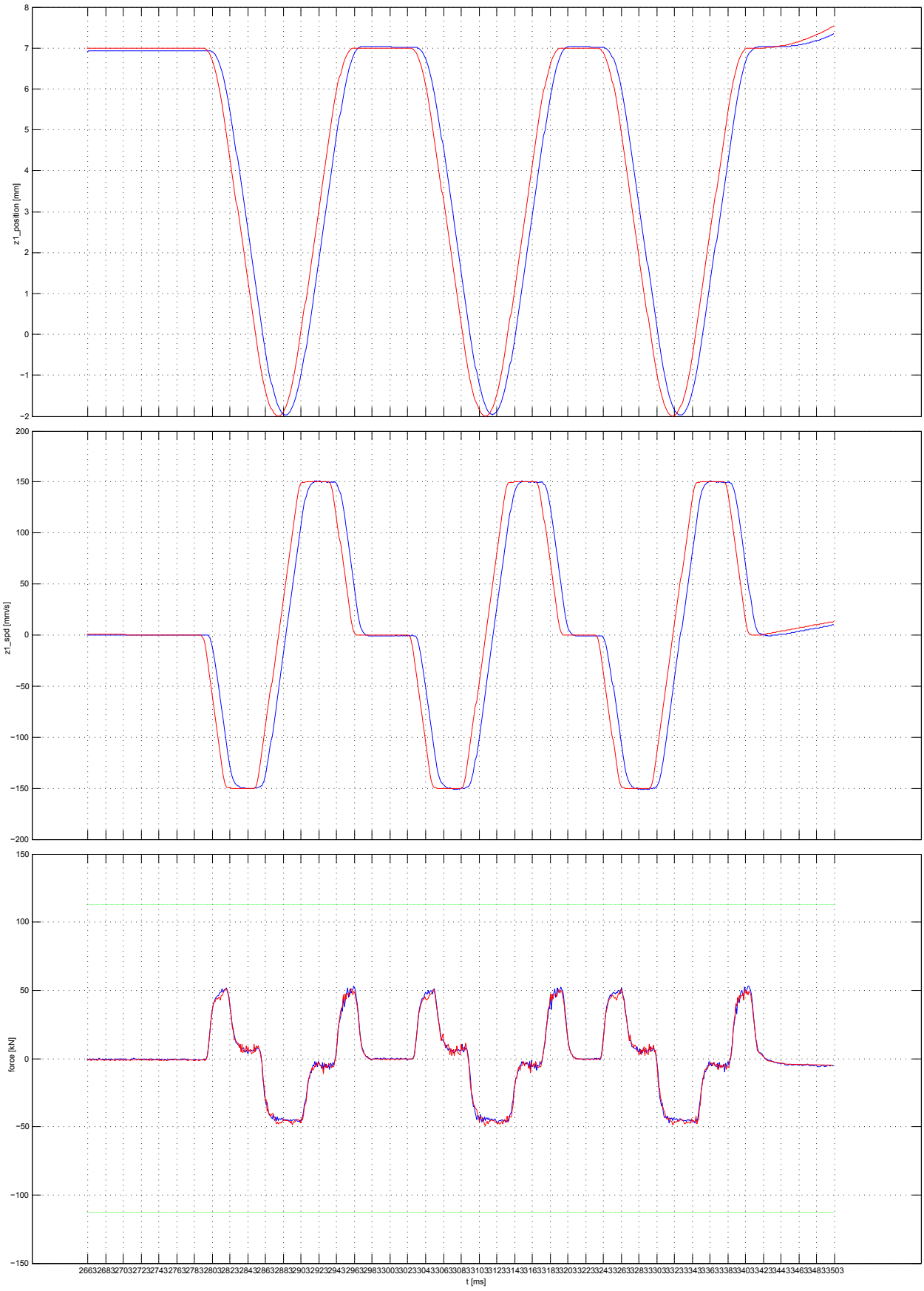


Figure 80: Punch pattern without the steel plate

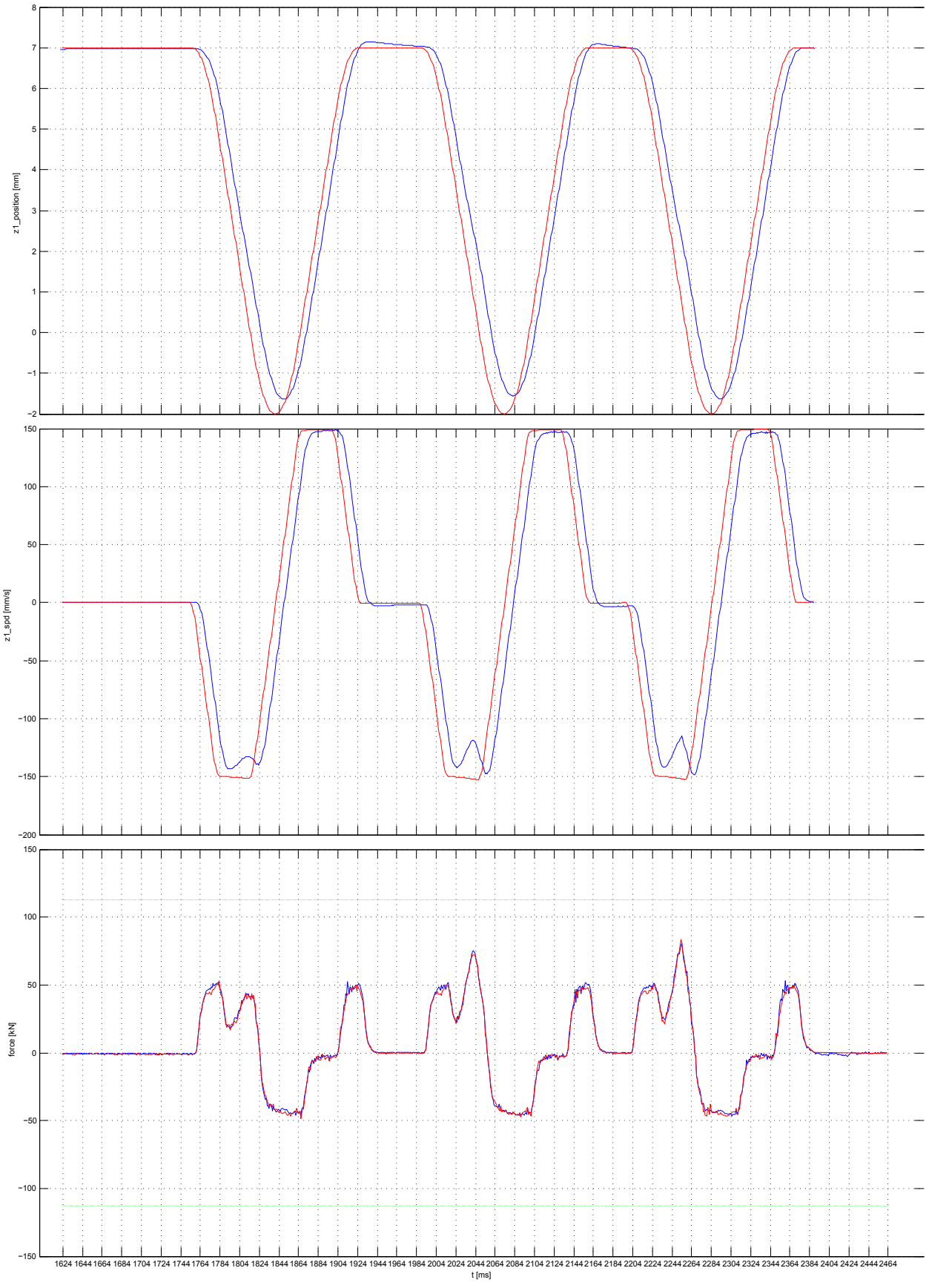


Figure 81: Punch pattern with the 5 mm steel plate

Appendix D Source code

D.1 pattern_stm.c

```
#include "../../include/touch_rt.h"

#include <rtai.h>
#include <rtai_sched.h>
#include <rtai_shm.h>
#include <math.h>

#include "pattern_stm.h"
#include "restart_stm.h"

#include "../../stm/stm_mod.h"
#include "../../servio/servio_mod.h"
#include "../../axes/axes_mod.h"
#include "../../tgen/tgen_mod.h"
#include "../../ldp/ldp_mod.h"
#include "../../ldp/ldp_cmd.h"
#include "../../lut/lut_mod.h"

#define DEBUG_TAG "pattern"
#include "../../debug/debug_mod.h"

#include "../../include/gendef.h"
#include "../../include/axdef.h"

//state functions prototypes
void Pattern_s_Idle();
void Pattern_s_Init();
void Pattern_s_Referencing();
void Pattern_s_After_Referencing();
void Pattern_s_Z1_In_Pos();
void Pattern_s_X1_In_Pos();
void Pattern_s_Punching();
void Pattern_s_Moving();
void Pattern_s_Anticipating_X1();
void Pattern_s_Anticipating_Z1();
void Pattern_s_Z1_Moving_Up();
void Pattern_s_Z2_disabling();
void Pattern_s_Punching_2();

//condition function prototypes
int Pattern_c_AlwaysTrue();
int Pattern_c_NewCommand();
int Pattern_c_Z1_Ready();
int Pattern_c_Z2_Ready();
int Pattern_c_X1_Ready();
int Pattern_c_Z1_In_Pos();
int Pattern_c_X1_In_Pos();
int Pattern_c_Referenced();
int Pattern_c_Counter();
int Pattern_c_Z1_Above_Clear_Height();
int Pattern_c_Time_To_Punch();
int Pattern_c_Time_To_Move();
int Pattern_c_Z1_Goes_Up();
int Pattern_c_Z2_disabled();
int Pattern_c_Z2_enabled();
int Pattern_c_timer_elapsed();
int Pattern_c_timer_2_elapsed();

//transition function prototypes
void Pattern_t_DoNothing();
void Pattern_t_Enable_Z2();
void Pattern_t_Disable_Z2();
```

```
63  void Pattern_t_IdleToInit();
64  void Pattern_t_InitToRef();
65  void Pattern_t_RefToX1Pos();
66  void Pattern_t_X1PosToZ1Pos();
67  void Pattern_t_PunchToAnticipating_Z1();
68  void Pattern_t_Anticipating_Z1ToMoving();
69  void Pattern_t_MovingToAnticipating_X1();
70  void Pattern_t_Anticipating_X1ToPunch();
71  void Pattern_t_PunchToPunch();
72  void Pattern_t_PunchToZ1up();
73  void Pattern_t_PunchToIdle();
74
75  //global variables
76  ttv_recv_cmd_data gtv_cmd_data_recv;
77  ttv_send_cmd_data gtv_cmd_data_send;
78
79  int giv_command = CMD_EVACUATE_TABLE;
80
81  // loaded from config.ini
82  float gfv_x1_axis_speed = 10;
83  float gfv_x1_axis_acc = 10;
84  float gfv_x1_axis_start_pos= 200; // axis X1 start position [mm]
85  float gfv_x1_axis_pos = 200; // axis X1 position [mm]
86  float gfv_x1_axis_delta = 1; // [mm] distance between two punches
87  int giv_x1_nr_punch_dir = 5; // number of one way movements
88
89  // loaded from config.ini
90  float gfv_z1_axis_start_pos = 50;
91  float gfv_z1_clear_height = 5; // mm distance above table
92  int giv_punch_nr = 1;
93  float gfv_time_to_position_up_correction = 0.0; //z1 time to position up correction [s]
94  float gfv_time_to_position_down_correction = 0.0; //z1 time to position correction [s]
95  float gfv_z1_axis_die_penetration = -1; //default z1 position for die penetration [mm]
96  float gfv_end_position = 56; // [mm] position reached when the punch pattern is done
97  float gfv_z1_ref_position = 56.3; // [mm] position after referencing
98
99  float gfv_punch_distance = 0.0;
100
101 int giv_move_direction = -1;
102 int giv_internal_move_count = 0;
103
104 // punch count
105 int giv_punch_count = 0;
106 // old punch count
107 int giv_pattern_moved=0;
108
109 //z1 time to position [s], axis goes from hower height to clear height
110 float gfv_z1_time_to_pos = 99.0;
111 //z1 time to position up [s], axis goes from die penetration height to clear height
112 float gfv_time_to_position_up = 99.0;
113
114 int pattern_stm_initialise()
115 {
116     debug_message(DEBUG_TAG,"pattern_stm_initialise");
117
118     int liv_stm_idx = stm_register_state_machine();
119     if (liv_stm_idx == -1)
120     {
121       debug_message(DEBUG_TAG,"stm_register_state_machine failed");
122       return -1;
123     }
124
125     int liv_state_idle = stm_register_state(liv_stm_idx,Pattern_s_Idle);
126     int liv_state_init = stm_register_state(liv_stm_idx,Pattern_s_Init);
127     int liv_state_ref = stm_register_state(liv_stm_idx,Pattern_s_Referencing);
128     int liv_state_after_ref = stm_register_state(liv_stm_idx,Pattern_s_After_Referencing);
129     int liv_state_z1_in_pos = stm_register_state(liv_stm_idx,Pattern_s_Z1_In_Pos);
130     int liv_state_x1_in_pos = stm_register_state(liv_stm_idx,Pattern_s_X1_In_Pos);
131     int liv_state_moving = stm_register_state(liv_stm_idx,Pattern_s_Moving);
132     int liv_state_anticipating_X1 = stm_register_state(liv_stm_idx,Pattern_s_Anticipating_X1);
```

```
133     int liv_state_punching = stm_register_state(liv_stm_idx,Pattern_s_Punching);
134     int liv_state_punching_2 = stm_register_state(liv_stm_idx,Pattern_s_Punching_2);
135     int liv_state_anticipating_Z1 = stm_register_state(liv_stm_idx,Pattern_s_Anticipating_Z1);
136     int liv_state_z1_moving_up = stm_register_state(liv_stm_idx,Pattern_s_Z1_Moving_Up);
137     int liv_state_disabling_z2 = stm_register_state(liv_stm_idx,Pattern_s_Z2_disabling);
138
139     if ((liv_state_idle == -1) ||
140         (liv_state_init == -1) ||
141         (liv_state_ref == -1) ||
142         (liv_state_z1_in_pos == -1) ||
143         (liv_state_x1_in_pos == -1) ||
144         (liv_state_punching == -1) ||
145         (liv_state_anticipating_Z1 == -1) ||
146         (liv_state_anticipating_X1 == -1) ||
147         (liv_state_moving == -1) ||
148         (liv_state_z1_moving_up == -1) ||
149         (liv_state_disabling_z2 == -1) ||
150         (liv_state_moving == -1))
151     {
152       debug_message(DEBUG_TAG,"stm_register_state failed");
153       return -1;
154     }
155
156     int liv_cond_true = stm_register_condition(liv_stm_idx,Pattern_c_AlwaysTrue);
157     int liv_cond_new_cmd = stm_register_condition(liv_stm_idx,Pattern_c_NewCommand);
158
159     int liv_cond_z2_disabled = stm_register_condition(liv_stm_idx,Pattern_c_Z2_disabled);
160     int liv_cond_z2_enabled = stm_register_condition(liv_stm_idx,Pattern_c_Z2_enabled);
161
162     int liv_cond_z1_rdy = stm_register_condition(liv_stm_idx,Pattern_c_Z1_Ready);
163     int liv_cond_z2_rdy = stm_register_condition(liv_stm_idx,Pattern_c_Z2_Ready);
164     int liv_cond_x1_rdy = stm_register_condition(liv_stm_idx,Pattern_c_X1_Ready);
165     int liv_cond_x1_rdy_z1_ready = stm_register_condition_combination(liv_stm_idx,STM_OPER_AND,liv_cond_z1_rdy
          ,liv_cond_x1_rdy);
166     int liv_cond_z1_in_pos = stm_register_condition(liv_stm_idx,Pattern_c_Z1_In_Pos);
167     int liv_cond_z1_not_in_pos = stm_register_condition_combination(liv_stm_idx,STM_OPER_NOT,
          liv_cond_z1_in_pos,0);
168     int liv_cond_x1_in_pos = stm_register_condition(liv_stm_idx,Pattern_c_X1_In_Pos);
169     int liv_cond_x1_not_in_pos = stm_register_condition_combination(liv_stm_idx,STM_OPER_NOT,
          liv_cond_x1_in_pos,0);
170     int liv_cond_x1_rdy_not_in_pos = stm_register_condition_combination(liv_stm_idx,STM_OPER_AND,
          liv_cond_x1_not_in_pos,liv_cond_x1_rdy);
171     int liv_cond_x1_rdy_in_pos = stm_register_condition_combination(liv_stm_idx,STM_OPER_AND,
          liv_cond_x1_in_pos,liv_cond_x1_rdy);
172     int liv_cond_time_to_move = stm_register_condition(liv_stm_idx,Pattern_c_Time_To_Move);
173     int liv_cond_x1_time_to_punch = stm_register_condition(liv_stm_idx,Pattern_c_Time_To_Punch);
174     int liv_cond_x1_rdy_z1_in_pos = stm_register_condition_combination(liv_stm_idx,STM_OPER_AND,
          liv_cond_z1_in_pos,liv_cond_x1_rdy);
175     int liv_cond_x1_rdy_z1_not_in_pos = stm_register_condition_combination(liv_stm_idx,STM_OPER_AND,
          liv_cond_z1_not_in_pos,liv_cond_x1_rdy);
176     int liv_cond_ref = stm_register_condition(liv_stm_idx,Pattern_c_Referenced);
177     int liv_cond_not_ref = stm_register_condition_combination(liv_stm_idx,STM_OPER_NOT,liv_cond_ref,0);
178     int liv_cond_counter = stm_register_condition(liv_stm_idx,Pattern_c_Counter);
179     int liv_cond_not_counter = stm_register_condition_combination(liv_stm_idx,STM_OPER_NOT,liv_cond_counter,0)
          ;
180     int liv_cond_x1_time_to_punch_counter = stm_register_condition_combination(liv_stm_idx,STM_OPER_AND,
          liv_cond_x1_time_to_punch,liv_cond_counter);
181     int liv_cond_not_counter_z1_x1_ready = stm_register_condition_combination(liv_stm_idx,STM_OPER_AND,
          liv_cond_x1_rdy_z1_ready,liv_cond_not_counter);
182     int liv_cond_not_counter_z1_x1_z2_ready = stm_register_condition_combination(liv_stm_idx,STM_OPER_AND,
          liv_cond_not_counter_z1_x1_ready,liv_cond_z2_rdy);
183     int liv_cond_x1_time_to_punch_z1_ready = stm_register_condition_combination(liv_stm_idx,STM_OPER_AND,
          liv_cond_z1_rdy,liv_cond_x1_time_to_punch);
184     int liv_cond_liv_cond_time_to_move_and_x1_ready = stm_register_condition_combination(liv_stm_idx,
          STM_OPER_AND,liv_cond_time_to_move,liv_cond_x1_rdy);
185
186     int liv_cond_not_counter_z1_x1_ready_z2_dis = stm_register_condition_combination(liv_stm_idx,STM_OPER_AND,
          liv_cond_not_counter_z1_x1_ready,liv_cond_z2_disabled);
187     int liv_cond_x1_rdy_z1_not_in_pos_z2_disabled = stm_register_condition_combination(liv_stm_idx,
          STM_OPER_AND,liv_cond_x1_rdy_z1_not_in_pos,liv_cond_z2_disabled);
```

```
188   int liv_cond_x1_time_to_punch_z1_ready_z2_enabled = stm_register_condition_combination(liv_stm_idx,
          STM_OPER_AND,liv_cond_x1_time_to_punch_z1_ready,liv_cond_z2_enabled);
189   int liv_cond_not_ref_z2_disabled = stm_register_condition_combination(liv_stm_idx,STM_OPER_AND,
          liv_cond_not_ref,liv_cond_z2_disabled);
190   int liv_cond_timer_elapsed = stm_register_condition(liv_stm_idx, Pattern_c_timer_elapsed);
191   int liv_cond_timer_2_elapsed = stm_register_condition(liv_stm_idx, Pattern_c_timer_2_elapsed);
192   int liv_cond_x1_time_to_punch_z1_ready_z2_enabled_timer = stm_register_condition_combination(liv_stm_idx,
          STM_OPER_AND,liv_cond_x1_time_to_punch_z1_ready_z2_enabled,liv_cond_timer_elapsed);
193
194   if ((liv_cond_true == -1) ||
195       (liv_cond_new_cmd == -1) ||
196       (liv_cond_z1_rdy == -1) ||
197       (liv_cond_x1_rdy == -1) ||
198       (liv_cond_z1_in_pos == -1) ||
199       (liv_cond_z1_not_in_pos == -1) ||
200       (liv_cond_x1_rdy_in_pos == -1) ||
201       (liv_cond_x1_rdy_not_in_pos == -1) ||
202       (liv_cond_x1_in_pos == -1) ||
203       (liv_cond_z2_rdy == -1) ||
204       (liv_cond_z2_enabled == -1) ||
205       (liv_cond_z2_disabled == -1) ||
206       (liv_cond_not_counter_z1_x1_ready_z2_dis == -1) ||
207       (liv_cond_x1_not_in_pos == -1) ||
208       (liv_cond_x1_rdy_z1_in_pos == -1) ||
209       (liv_cond_x1_rdy_z1_not_in_pos == -1) ||
210       (liv_cond_ref == -1) ||
211       (liv_cond_not_ref == -1) ||
212       (liv_cond_counter == -1) ||
213       (liv_cond_time_to_move == -1) ||
214       (liv_cond_x1_time_to_punch == -1) ||
215       (liv_cond_not_counter_z1_x1_z2_ready == -1)||
216       (liv_cond_x1_time_to_punch_counter == -1) ||
217       (liv_cond_x1_time_to_punch_z1_ready == -1) ||
218       (liv_cond_liv_cond_time_to_move_and_x1_ready == -1) ||
219       (liv_cond_x1_rdy_z1_not_in_pos_z2_disabled == -1) ||
220       (liv_cond_x1_time_to_punch_z1_ready_z2_enabled == -1) ||
221       (liv_cond_not_ref_z2_disabled == -1) ||
222       (liv_cond_x1_rdy_z1_ready == -1) ||
223       (liv_cond_timer_2_elapsed == -1))
224   {
225     debug_message(DEBUG_TAG,"stm_register_condition failed");
226     return -1;
227   }
228
229   int liv_trans_1 = stm_register_transition(liv_stm_idx,liv_state_idle,liv_state_init,liv_cond_new_cmd,
          Pattern_t_IdleToInit);
230   int liv_trans_2 = stm_register_transition(liv_stm_idx,liv_state_init,liv_state_after_ref,liv_cond_ref,
          Pattern_t_DoNothing);
231   int liv_trans_3 = stm_register_transition(liv_stm_idx,liv_state_init,liv_state_ref,
          liv_cond_not_ref_z2_disabled,Pattern_t_InitToRef);
232   int liv_trans_3_1 = stm_register_transition(liv_stm_idx,liv_state_ref,liv_state_after_ref,liv_cond_ref,
          Pattern_t_DoNothing);
233
234   int liv_trans_4 = stm_register_transition(liv_stm_idx,liv_state_after_ref,liv_state_x1_in_pos,
          liv_cond_x1_rdy_in_pos,Pattern_t_Disable_Z2);
235   int liv_trans_5 = stm_register_transition(liv_stm_idx,liv_state_after_ref,liv_state_x1_in_pos,
          liv_cond_x1_rdy_not_in_pos,Pattern_t_RefToX1Pos);
236   int liv_trans_6 = stm_register_transition(liv_stm_idx,liv_state_x1_in_pos,liv_state_z1_in_pos,
          liv_cond_x1_rdy_z1_in_pos,Pattern_t_DoNothing);
237   int liv_trans_7 = stm_register_transition(liv_stm_idx,liv_state_x1_in_pos,liv_state_z1_in_pos,
          liv_cond_x1_rdy_z1_not_in_pos,Pattern_t_X1PosToZ1Pos);
238   int liv_trans_8 = stm_register_transition(liv_stm_idx,liv_state_z1_in_pos,liv_state_moving,
          liv_cond_x1_rdy_z1_ready,Pattern_t_Enable_Z2);
239
240   int liv_trans_9 = stm_register_transition(liv_stm_idx,liv_state_anticipating_Z1,liv_state_moving,
          liv_cond_liv_cond_time_to_move_and_x1_ready,Pattern_t_Anticipating_Z1ToMoving);
241   int liv_trans_10 = stm_register_transition(liv_stm_idx,liv_state_moving,liv_state_punching,
          liv_cond_x1_time_to_punch_z1_ready_z2_enabled_timer,Pattern_t_Anticipating_X1ToPunch);
242   int liv_trans_15 = stm_register_transition(liv_stm_idx,liv_state_punching,liv_state_punching_2,
          liv_cond_timer_2_elapsed,Pattern_t_PunchToPunch);
```

```
243     int liv_trans_12 = stm_register_transition(liv_stm_idx,liv_state_punching_2,liv_state_anticipating_Z1,
            liv_cond_counter,Pattern_t_PunchToAnticipating_Z1);
244
245     int liv_trans_13_0 = stm_register_transition(liv_stm_idx,liv_state_punching_2,liv_state_disabling_z2,
            liv_cond_not_counter_z1_x1_z2_ready,Pattern_t_Disable_Z2);
246     int liv_trans_13 = stm_register_transition(liv_stm_idx,liv_state_disabling_z2,liv_state_z1_moving_up,
            liv_cond_z2_disabled,Pattern_t_PunchToZ1up);
247     int liv_trans_14 = stm_register_transition(liv_stm_idx,liv_state_z1_moving_up,liv_state_idle,
            liv_cond_z1_rdy,Pattern_t_PunchToIdle);
248
249     if ((liv_trans_1 == -1) ||
250         (liv_trans_2 == -1) ||
251         (liv_trans_3 == -1) ||
252         (liv_trans_3_1 == -1) ||
253         (liv_trans_4 == -1) ||
254         (liv_trans_5 == -1) ||
255         (liv_trans_6 == -1) ||
256         (liv_trans_7 == -1) ||
257         (liv_trans_8 == -1) ||
258         (liv_trans_9 == -1) ||
259         (liv_trans_10 == -1) ||
260         (liv_trans_12 == -1) ||
261         (liv_trans_14 == -1) ||
262         (liv_trans_13_0 == -1) ||
263         (liv_trans_15 == -1) ||
264         (liv_trans_13 == -1))
265     {
266       debug_message(DEBUG_TAG,"stm_register_transition failed");
267       return -1;
268     }
269
270     return 1;
271 }
272
273
274 //state functions
275 void Pattern_s_Idle()
276 {
277 }
278
279 void Pattern_s_Init()
280 {
281 }
282
283 void Pattern_s_Referencing()
284 {
285 }
286
287 void Pattern_s_After_Referencing()
288 {
289 }
290
291 void Pattern_s_Z1_In_Pos()
292 {
293 }
294
295 void Pattern_s_X1_In_Pos()
296 {
297 }
298
299 void Pattern_s_Punching()
300 {
301 }
302
303 void Pattern_s_Punching_2()
304 {}
305
306 void Pattern_s_Moving()
307 {
308 }
```

```
309
310 void Pattern_s_Anticipating_X1()
311 {
312 }
313 void Pattern_s_Anticipating_Z1()
314 {
315 }
316
317 void Pattern_s_Z1_Moving_Up()
318 {
319 }
320
321 void Pattern_s_Z2_disabling()
322 {
323 }
324
325 //conditions functions
326 int Pattern_c_AlwaysTrue()
327 {
328    return 1;
329 }
330
331 int Pattern_c_NewCommand()
332 {
333   return ldp_cmd_ready(giv_command);
334 }
335
336 int Pattern_c_Z1_Ready()
337 {
338    return (axis_get_state(Z1_AXIS) == ST_READY);
339 }
340
341 int Pattern_c_Z2_Ready()
342 {
343    return (axis_get_state(Z2_AXIS) == ST_READY);
344 }
345
346 int Pattern_c_X1_Ready()
347 {
348    return (axis_get_state(X1_AXIS) == ST_READY);
349 }
350
351 int Pattern_c_Z1_In_Pos()
352 {
353   return ( fabs(axis_get_position(Z1_AXIS)-gfv_z1_axis_start_pos)<0.001);
354 }
355
356 int Pattern_c_X1_In_Pos()
357 {
358    return ( fabs(axis_get_position(X1_AXIS)-gfv_x1_axis_start_pos)<0.001);
359 }
360
361 int Pattern_c_Z1_Above_Clear_Height()
362 {
363    return (axis_get_req_position(Z1_AXIS)>=gfv_z1_clear_height);
364 }
365
366 int Pattern_c_Referenced()
367 {
368    return restart_is_done();
369 }
370
371 int Pattern_c_Counter()
372 {
373    return (giv_punch_nr > giv_punch_count);
374 }
375
376 int Pattern_c_Time_To_Punch()
377 {
378    float pfv_time_to_move_ready = axis_get_time_to_move_ready(X1_AXIS);
```

```
379     float pfv_time_to_pos = gfv_z1_time_to_pos+gfv_time_to_position_down_correction;
380     if((pfv_time_to_move_ready <= pfv_time_to_pos))//&& lut_timer_elapsed(1))
381     {
382       return 1;
383     }
384     return 0;
385 
386 }
387 
388 int Pattern_c_Time_To_Move()
389 {
390     float pfv_act_time = axis_get_act_time(Z1_AXIS);
391     if((pfv_act_time >= gfv_time_to_position_up+gfv_time_to_position_up_correction))//&& lut_timer_elapsed(1))
392     {
393       return 1;
394     }
395     return 0;
396 }
397 
398 int Pattern_c_Z1_Goes_Up()
399 {
400     return ((0.1 <= axis_get_req_speed(Z1_AXIS)) || (axis_get_state(Z1_AXIS) == ST_READY)) ;
401 }
402 
403 int Pattern_c_Z2_disabled()
404 {
405    return ((~axis_is_enabled(Z2_AXIS)) | (axis_get_state(Z2_AXIS) == ST_IDLE));
406 }
407 
408 int Pattern_c_Z2_enabled()
409 {
410    return axis_is_enabled(Z2_AXIS);
411 }
412 
413 int Pattern_c_timer_elapsed()
414 {
415     return lut_timer_elapsed(1);
416 }
417 
418 int Pattern_c_timer_2_elapsed()
419 {
420     return lut_timer_elapsed(2);
421 }
422 
423 //transition functions
424 void Pattern_t_DoNothing()
425 {
426     debug_message(DEBUG_TAG,"Pattern_t_DoNothing");
427 }
428 
429 void Pattern_t_Enable_Z2()
430 {
431     debug_message(DEBUG_TAG,"Pattern_t_Enable_Z2");
432     lut_timer_set(1,0.2);
433     enable_axis(Z2_AXIS);
434     axis_set_control_flag(1); // axis enabled, set act and req position same as Z1 axis
435 }
436 
437 void Pattern_t_Disable_Z2()
438 {
439     debug_message(DEBUG_TAG,"Pattern_t_Disable_Z2");
440     disable_axis(Z2_AXIS);
441 }
442 
443 void Pattern_t_IdleToInit()
444 {
445     debug_message(DEBUG_TAG,"Pattern_IdleToInit");
446     gfv_x1_axis_speed = axis_get_param(X1_AXIS, 28);
447     gfv_x1_axis_acc = axis_get_param(X1_AXIS, 29);
448     gfv_x1_axis_delta = axis_get_param(X1_AXIS, 30); // distance between strokes [mm]
```

```
449     gfv_x1_axis_start_pos = axis_get_param(X1_AXIS, 40); // [mm]
450     giv_x1_nr_punch_dir = axis_get_param(X1_AXIS, 31); // number of one way movements
451     gfv_x1_axis_pos = gfv_x1_axis_start_pos;
452
453     gfv_z1_axis_start_pos = axis_get_param(Z1_AXIS, 40);
454     gfv_z1_axis_die_penetration = -1 * axis_get_param(Z1_AXIS, 26);
455     gfv_z1_clear_height = axis_get_param(Z1_AXIS, 29);
456
457     gfv_time_to_position_down_correction = axis_get_param(Z1_AXIS,28);
458     gfv_time_to_position_up_correction = axis_get_param(Z1_AXIS,27);
459     giv_punch_nr = axis_get_param(Z1_AXIS, 30);
460     giv_punch_count = 0;
461     giv_move_direction = -1;
462
463     axis_set_control_flag(0); // axis disabled, set point is beeing set
464
465     gfv_punch_distance = gfv_z1_axis_start_pos - gfv_z1_axis_die_penetration;
466
467     disable_axis(Z2_AXIS);
468     ldp_cmd_read(giv_command,&gtv_cmd_data_recv);
469 }
470
471 void Pattern_t_InitToRef()
472 {
473     debug_message(DEBUG_TAG,"Pattern_t_InitToRef");
474     restart_set_restart_request();
475 }
476
477 void Pattern_t_RefToX1Pos()
478 {
479     debug_message(DEBUG_TAG,"Pattern_t_RefToX1Pos");
480     axis_move_to_pos_w_spd_acc(X1_AXIS,gfv_x1_axis_start_pos,200,160);
481     disable_axis(Z2_AXIS);
482 }
483
484 void Pattern_t_X1PosToZ1Pos()
485 {
486     debug_message(DEBUG_TAG,"Pattern_t_X1PosToZ1Pos");
487     axis_move_to_pos_w_spd_acc(Z1_AXIS,gfv_z1_axis_start_pos,80,150);
488 }
489
490 void Pattern_t_PunchToAnticipating_Z1()
491 {
492     debug_message(DEBUG_TAG,"Pattern_t_PunchToAnticipating_Z1");
493     // calculate times to travel through hover height
494     gfv_z1_time_to_pos = axis_get_time_to_position(Z1_AXIS,gfv_z1_axis_start_pos - gfv_z1_clear_height);
495     // calculate time to travel back to maximal hover height
496     gfv_time_to_position_up = axis_get_time_to_position_up(Z1_AXIS,gfv_z1_axis_start_pos - gfv_z1_clear_height
            );
497 }
498
499
500 void Pattern_t_Anticipating_Z1ToMoving()
501 {
502     debug_message(DEBUG_TAG,"Pattern_t_Anticipating_Z1ToMoving");
503
504     if(giv_punch_nr > giv_x1_nr_punch_dir)
505     {
506       if(giv_internal_move_count < (giv_x1_nr_punch_dir))
507       {
508         gfv_x1_axis_pos+=giv_move_direction * gfv_x1_axis_delta;
509         giv_internal_move_count++;
510       }
511       else
512       {
513         gfv_x1_axis_pos += giv_move_direction * gfv_x1_axis_delta / 2;
514         giv_move_direction = -1 * giv_move_direction;
515         giv_internal_move_count = 0;
516       }
517     }
```

```
518     else
519     {
520       int piv_punch_half_nr = giv_punch_nr/2;
521       if (2*piv_punch_half_nr == giv_punch_nr)
522       {
523         if(giv_punch_count<piv_punch_half_nr)
524             gfv_x1_axis_pos-=gfv_x1_axis_delta;
525         if(giv_punch_count==piv_punch_half_nr)
526             gfv_x1_axis_pos+=gfv_x1_axis_delta/2;
527         if(giv_punch_count>piv_punch_half_nr)
528             gfv_x1_axis_pos+=gfv_x1_axis_delta;
529       }
530       else
531       {
532         if(giv_punch_count<piv_punch_half_nr+1)
533             gfv_x1_axis_pos-=gfv_x1_axis_delta;
534         if(giv_punch_count==piv_punch_half_nr+1)
535             gfv_x1_axis_pos+=gfv_x1_axis_delta/2;
536         if(giv_punch_count>piv_punch_half_nr+1)
537             gfv_x1_axis_pos+=gfv_x1_axis_delta;
538       }
539     }
540     axis_move_to_pos_w_spd_acc(X1_AXIS,gfv_x1_axis_pos,gfv_x1_axis_speed,gfv_x1_axis_acc);
541 }
542
543
544 void Pattern_t_MovingToAnticipating_X1()
545 {
546     debug_message(DEBUG_TAG,"Pattern_t_MovingToAnticipating_X1");
547 }
548
549 void Pattern_t_Anticipating_X1ToPunch()
550 {
551     debug_message(DEBUG_TAG,"Pattern_t_Anticipating_X1ToPunch");
552
553     giv_punch_count++;
554
555     axis_pos_loop_punch_cycle_with_const_speed_zone(Z1_AXIS, gfv_punch_distance);
556     axis_pos_loop_punch_cycle_with_const_speed_zone(Z2_AXIS, gfv_punch_distance);
557     axis_set_control_flag(2); // axis is not disabled between punches and position loop is executed
558 }
559
560 void Pattern_t_PunchToZ1up()
561 {
562     debug_message(DEBUG_TAG,"Pattern_t_PunchToZ1up");
563     axis_move_to_pos_w_spd_acc(Z1_AXIS,gfv_end_position,80,150);
564     axis_set_control_flag(0); // axis is disabled
565 }
566
567 void Pattern_t_PunchToIdle()
568 {
569     debug_message(DEBUG_TAG,"Pattern_t_PunchToIdle");
570     axis_set_req_pos(Z2_AXIS,gfv_end_position);
571     giv_punch_count = 0;
572     giv_move_direction = -1;
573     giv_internal_move_count = 0;
574     gtv_cmd_data_send.siv_ok = 1;
575     gtv_cmd_data_send.siv_cmd = gtv_cmd_data_recv.siv_cmd;
576     gtv_cmd_data_send.siv_sn = gtv_cmd_data_recv.siv_sn;
577     ldp_cmd_done(&gtv_cmd_data_send);
578 }
```

D.2 tgen_get_time_to_position_const_speed.c

```
float tgen_get_time_to_position_const_speed(int piv_idx,float pfv_position)
{
    float j = gtp_tgen_data->channels[piv_idx].init_data.sfv_jerk/1000.0;
    float a = gtp_tgen_data->channels[piv_idx].init_data.sfv_acc/1000.0;
    float v = gtp_tgen_data->channels[piv_idx].init_data.sfv_speed/1000.0;

    float distance = gtp_tgen_data->channels[piv_idx].init_data.sfv_a_a/1000; //p->init_data.sfv_a_a/1000.0;

    //time zone 1 - build up acc, limit jerk
        float t1 = a/j;
        //float a1 = a;
        float v1 = j*t1*t1/2.0;
        float s1 = j*t1*t1*t1/6.0;

        if(v1>=v/2) // max. speed will be reached during building up acc
    {
      v1 = v/2;
      t1 = sqrtf(2 * v1 / j);
      a = j * t1;
      s1 = j*t1*t1*t1/6.0;
    }

        //time zone 2 - max and constant acceleration
        float t2 = t1 + (v - 2*v1)/a;
        //float a2 = a;
        float v2 = v1 + a*(t2 - t1);
        float s2 = s1 + v1*(t2 - t1) + a*(t2 - t1)*(t2 - t1)/2.0;

        //time zone 3 - acceleration decreases as speed gets higher
        float t3 = t2 + a/j;
    if(v1>=v/2) // max. speed will be reached during building up acc
    {
       t3 = t2 + t1;
    }
        //float a3 = 0;
        float v3 = v2 + a*(t3 - t2) - j*(t3 - t2)*(t3 - t2)/2.0;
        float s3 = s2 + v2*(t3 - t2) + a*(t3 - t2)*(t3 - t2)/2.0 - j*(t3 - t2)*(t3 - t2)*(t3 - t2)/6.0;

        //time zone 4 - constant speed
        float t4 = t3 + 0.0;
        //float a4 = 0;
        float v4 = v;
        float s4 = s3 + (t4 - t3)*v;

        //time zone 5 - build up dec, limit jerk
        float t5 = t4 + a/j;
        //float a5 = -j*(t5 - t4);
        float v5 = v4 - j*(t5 - t4)*(t5 - t4)/2.0;
        float s5 = s4 + v4*(t5 - t4) - j*(t5 - t4)*(t5 - t4)*(t5 - t4)/6.0;

        //time zone 6 - max and constant deceleration
        float t6 = t5 + v5/a;
        //float a6 = -a;
        float v6 = v5 - a*(t6 - t5);
        float s6 = s5 + v5*(t6 - t5) - a*(t6 - t5)*(t6 - t5)/2.0;

    if(s6<distance)
    {
        debug_message(DEBUG_TAG,"Build constant speed zone");
        float distance_missing = distance - s6;

        //time zone 4 - constant speed
        t4 = t3 + distance_missing / v4;;
        s4 = s3 + (t4 - t3)*v;

        //time zone 5 - build up dec, limit jerk
        t5 = t4 + a/j;
```

```
68          //float a5 = -j*(t5 - t4);
69          v5 = v4 - j*(t5 - t4)*(t5 - t4)/2.0;
70          s5 = s4 + v4*(t5 - t4) - j*(t5 - t4)*(t5 - t4)*(t5 - t4)/6.0;
71
72          //time zone 6 - max and constant deceleration
73          t6 = t5 + v5/a;
74          //float a6 = -a;
75          v6 = v5 - a*(t6 - t5);
76          s6 = s5 + v5*(t6 - t5) - a*(t6 - t5)*(t6 - t5)/2.0;
77      }
78
79      if(s6 > distance)
80      {
81          debug_message(DEBUG_TAG,"Shortening const acc phase");
82
83          //float real_s2 = s2 - s1;
84          float ss3 = s3 - s2;
85          // desired s2
86          float desired_s2 = (distance - s1 - ss3)/2;
87
88
89          //time zone 2 - max and constant acceleration
90            //t2 = t1 + (v - 2*v1)/a;
91          float c[5]={0};
92          c[0] = - desired_s2;
93          c[1] = v1;
94          c[2] = 0.5 * a;
95          float tt2 = tgen_newton(c, 2, 1, 0, 9999); //tgen_newton(float c[5], int power, float start, float max,
                float min)
96
97          t2 = t1 + tt2;
98          v2 = v1 + a*(t2 - t1);
99          s2 = s1 + v1*(t2 - t1) + a*(t2 - t1)*(t2 - t1)/2.0;
100
101           //time zone 3 - acceleration decreases as speed gets higher
102           t3 = t2 + a/j;
103         if(v1>=v/2) // max. speed will be reached during building up acc
104         {
105               t3 = t2 + t1;
106         }
107           // a3 = 0;
108           v3 = v2 + a*(t3 - t2) - j*(t3 - t2)*(t3 - t2)/2.0;
109           v = v3;
110           s3 = s2 + v2*(t3 - t2) + a*(t3 - t2)*(t3 - t2)/2.0 - j*(t3 - t2)*(t3 - t2)*(t3 - t2)/6.0;
111
112           //time zone 4 - constant speed
113           t4 = t3 + 0.0;
114           //float a4 = 0;
115           v4 = v3;
116           s4 = s3 + (t4 - t3)*v;
117
118           //time zone 5 - build up dec, limit jerk
119           t5 = t4 + a/j;
120           //float a5 = -j*(t5 - t4);
121           v5 = v4 - j*(t5 - t4)*(t5 - t4)/2.0;
122           s5 = s4 + v4*(t5 - t4) - j*(t5 - t4)*(t5 - t4)*(t5 - t4)/6.0;
123
124           //time zone 6 - max and constant deceleration
125           t6 = v5/a + t5;
126           //float a6 = -a;
127           v6 = v5 - a*(t6 - t5);
128           s6 = s5 + v5*(t6 - t5) - a*(t6 - t5)*(t6 - t5)/2.0;
129     }
130
131     float ss[7] = {0};
132     ss[1]=s1;
133     ss[2]=s2;
134     ss[3]=s3;
135     ss[4]=s4;
136     ss[5]=s5;
```

```
137     ss[6]=s6;
138
139     float tt[7] = {0};
140     tt[1]=t1;
141     tt[2]=t2;
142     tt[3]=t3;
143     tt[4]=t4;
144     tt[5]=t5;
145     tt[6]=t6;
146
147     pfv_position /= 1000;
148     if(pfv_position>ss[6])
149     {
150       debug_float(DEBUG_TAG, "Desired position is further than maximal reachable position, time_to_pos
              calculation terminated. Returns -1. Position requested: ",ss[6]);
151       return 0;
152     }
153
154     // Detects first incomplete time zone - index i
155     int i=1;
156
157     float pfv_time_A = 0.0;
158     float pfv_pos_B = 0.0;
159     float pfv_time_B = 0.0;
160
161     do
162     {
163       if(pfv_position < ss[i])
164       {
165           //debug_message(DEBUG_TAG,"Podminka");
166           break;
167       }
168       // Time is equal of duration till last complete zone
169       if((pfv_position == ss[i]))
170       {
171           pfv_time_A = tt[i];
172           //debug_message(DEBUG_TAG,"Positions are equal.");
173           return pfv_time_A;
174       }
175       //debug_message(DEBUG_TAG,"Pricteni");
176       i++;
177     } while(i<=6);//(sizeof(p->s)/sizeof(p->s[s[0])))
178
179     // Calculates time
180     switch (i)
181     {
182       case 1:
183       {
184         pfv_time_B = pow(pfv_position/j*6.0,1.0/3.0);
185         pfv_pos_B = j*pfv_time_B*pfv_time_B*pfv_time_B/6.0;
186         break;
187       }
188       case 2:
189       {
190         float c[3] = {0};
191         c[0] = s1 - pfv_position; //c
192         c[1] = v1; //b
193         c[2] = a/2.0; //a
194
195         float x1 = (-c[1] + sqrtf( (c[1]*c[1]) - (4.0*c[2]) * (c[0]) ) ) / (2*c[2]);
196         float x2 = (-c[1] - sqrtf( (c[1]*c[1]) - (4.0*c[2]) * (c[0]) ) ) / (2*c[2]);
197
198         if(x1>=0)
199         {
200           pfv_time_B = x1;
201         } else
202         {
203           pfv_time_B = x2;
204         }
205         pfv_pos_B = s1 + v1*pfv_time_B + a*pfv_time_B*pfv_time_B/2.0;
```

```
206         break;
207       }
208       case 3:
209       {
210         float c[4] = {0};
211         c[0] = s2 - pfv_position;
212         c[1] = v2;
213         c[2] = a/2.0;
214         c[3] = -j/6.0;
215         pfv_time_B = tgen_newton(c,3,0.0001,t3,t2);
216         pfv_pos_B = s2 + v2*pfv_time_B + a*pfv_time_B*pfv_time_B/2.0 - j*pfv_time_B*pfv_time_B*pfv_time_B/6.0;
217         break;
218       }
219       case 4:
220       {
221         pfv_time_B = (pfv_position-s3)/v;
222         pfv_pos_B = s3 + pfv_time_B*v;
223         break;
224       }
225       case 5:
226       {
227         float c[4] = {0};
228         c[0] = s4 - pfv_position;
229         c[1] = v4;
230         c[3] = j/6.0;
231         pfv_time_B = tgen_newton(c,3,0.0001,t5,t4);
232 
233         pfv_pos_B = s4 + v4*pfv_time_B - j*pfv_time_B*pfv_time_B*pfv_time_B/6.0;
234         break;
235       }
236       case 6:
237       {
238         float c[3] = {0};
239         c[0] = s5 - pfv_position; //c
240         c[1] = v5; //b
241         c[2] = -a/2.0; //a
242 
243         float x1 = (-c[1] + sqrtf( (c[1]*c[1]) - (4.0*c[2]) * (c[0]) ) ) / (2*c[2]);
244         float x2 = (-c[1] - sqrtf( (c[1]*c[1]) - (4.0*c[2]) * (c[0]) ) ) / (2*c[2]);
245 
246         if(x1>=0)
247         {
248           pfv_time_B = x1;
249         } else
250         {
251           pfv_time_B = x2;
252         }
253         pfv_pos_B = s5 + v5*pfv_time_B - a*pfv_time_B*pfv_time_B/2.0;
254         break;
255       }
256       case 0:
257       {
258         debug_float(DEBUG_TAG,"Error. Calculated time is replaced by t5",t5);
259         pfv_time_B = t5;
260       }
261     }
262 
263     if(i>1)
264        pfv_time_B+=tt[i-1];
265 
266     return pfv_time_B;
267   }
268 }
```

Appendix E Optimization

E.1 Optimization.m

```
clc; close all; hold off; format long;

% load target positions
position

% initialization
global kp Tmax
Tmax= 0.2 % simulation time
dT=0.001; % simulation step time
kp=800; % initial condition

t = 0:dT:Tmax-0.001;

x0=[kp]; % passing parameter for fminsearch
sim('position_controller_tune',Tmax); % run simulation
position_controller_tune % open simulation window

% plots initial and ideal curve
figure(1);
h1 = plot(Position.Time,Position.Data(:,1),'m','LineWidth',3);
hold on;
h2 = plot(Position.Time,Position.Data(:,2),'G','LineWidth',3);
hold on;

% starts minimalization
OPTIONS=optimset('LargeScale','off','MaxIter',10,'Display','iter');
x=fminsearch('Criterion',x0,OPTIONS);

% plots final result
kpopt=x(1)
sim('position_controller_tune',Tmax);
figure(1);
h3 = plot(Position.Time,Position.Data(:,1),'R','LineWidth',2);
grid on;
legend('initial setup','target','final result')

break;
```

E.2 Criterion.m

```
function f=Criterion(x)

global kp Tmax
kp=x(1)

% run simulation
sim('position_controller_tune',Tmax);

% plot progress
h0 = plot(Position.Time,Position.Data(:,1),'B','LineWidth',1);
hold on;
drawnow

% supress legend
hsAnno = get(h0, 'Annotation');
hsLegend = get(hsAnno, 'LegendInformation');
set(hsLegend, 'IconDisplayStyle', 'off');

% minimalization criterion
J.Data = J.Data.^2;
```

```
21
22 % increase the effect of resonance in the steady state
23 start = length(J.Data)-640;
24 endd = length(J.Data);
25 J.Data(start:endd) = J.Data(start:endd)*100; % error is multiplied
26 crit = sum(J.Data);
27
28 f=crit;
```