



TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

Master's Thesis

Development of a feature for the specification of a temporally variable massage force and integration into an existing system

executed for the purpose of obtaining the academic degree of

Diplom-Ingenieur

under the direction of

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Johann Wassermann
(E325/A4 Institute of Mechanics and Mechatronics
Measurement and Actuator Division)

Submitted to the Vienna University of Technology

Faculty of Mechanical and Industrial Engineering

by

Leonardo Rafael Abbate Sbaffoni B.Sc.

00929450
Wien

Vienna, June 2019

Leonardo Rafael Abbate Sbaffoni



TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

I acknowledge that I'm only allowed to the printing of my work under the classification of

Master's Thesis

only with approval of the examination board.

I declare that I have carried out my master's thesis independently and according to the recognized principles for scientific work and that all aids used, in particular the underlying literature, have been mentioned.

Furthermore, I declare that I have not presented this master's thesis subject either in Austria or abroad to an assessor for assessment or in any form as a thesis; and that this work is consistent with the one assessed by the reviewer.

Vienna, June 2019

Leonardo Rafael Abbate Scaffoni

Foreword

This master's thesis was written at the Institute of Mechanics and Mechatronics of the Vienna University of Technology to complete my studies in mechanical engineering under the direction of Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Johann Wassermann, whom I want to thank for assigning me to participate in this project and for all his help during the long debugging sessions.

I also want to thank the colleagues of the laboratory, Manfred Neumann and Reinhold Wagner for helping me with questions, assisting me when learning new things and helping fix the issues of the project.

Abstract

This master's thesis continues the development of a prototype for a therapeutic automatic massage bed to be used in hospitals for the regular treatment of bedridden patients. The automatic functions and programming capabilities of this device would allow to free up time of valuable therapists dedicated to routine treatments and allow for preemptive massage care in real world situations where medical personnel is just too overworked to perform it properly.

With the main core of the prototype being already developed (including user interface on the PC, preprocessing, fieldbus system, axes' controllers, sensorics and all their encompassing programming) the work performed for this thesis focused on improving the functionality of the vertical axis of the machine that applies the massage force to the patient, expanding its functionality to be able to perform massages with custom force profiles.

A new interface was programmed in LabVIEW that allows to draw new massage paths by hand using a pressure sensitive tablet by Wacom. This program stores the pressure values as the desired force values. The main program that runs in the PC and directs all the peripheral axis controllers was then modified to accept this new information and transmit it over to the controller of the vertical axis using the existing fieldbus infrastructure.

On the controller side, a feasibility analysis was performed to assess if the available processing power and memory of the existing microprocessor was adequate to the task. It was concluded that the microprocessor can still be adapted to perform this new function. The existing PID cascade controller was adapted and improved based on the concept of a switching position/force controller to follow a force setpoint. Additionally, many improvements were made on the handling of the measured process variable, the force, including a better zeroing logic and the addition of a programmatic nonlinear median filter to adapt to the existing design of the massage tool.

Kurzfassung

Diese Diplomarbeit setzt die Entwicklung eines Prototyps für ein therapeutisches, automatisches Massagebett fort. Dieser ist für die regelmäßige Behandlung von bettlägerigen Patienten in Krankenhäusern entwickelt worden. Die automatischen Funktionen und Programmiermöglichkeiten dieses Gerätes würde wertvolle Zeit von Therapeuten einsparen und zusätzlich den Einsatz von präventiver Medizin in Situationen erlauben, wo das Personal bereits zeitlich überlastet ist um diese ordnungsgemäß auszuüben.

Weil der Kern des Prototyps schon entwickelt worden ist (inkludierend der Benutzeroberfläche am PC, Vorbereitung, Feldbussystem, Achsenkontroller, Sensorik und der notwendigen Programmierung), fokussiert sich diese Arbeit auf die Verbesserung der Funktionalität der vertikalen Achse der Maschine, die die Kraft auf den Patienten ausübt. Die Funktionalität wurde erweitert, um Massagen mit personalisierten Kraftprofilen durchführen zu können.

Eine neue Benutzeroberfläche wurde in LabVIEW programmiert, die das händische Zeichnen von Massagepfaden mit einem drucksensitiven Tablet von Wacom erlaubt. Dieses Programm speichert die Druckwerte als die gewünschten Kraftwerte. Das Hauptprogramm, das am PC ausgeführt wird und die Achsenkontroller kontrolliert wurde modifiziert um diese neue Information zu akzeptieren und zum Kontroller der vertikalen Achse über die vorhandene Feldbusinfrastruktur zu übertragen.

Auf Seite des Kontrollers wurde eine Machbarkeitsanalyse durchgeführt, um die Angemessenheit des Prozessors und dessen Geschwindigkeit samt Speicherkapazität für die neue Funktionalitätserweiterung zu beurteilen. Diese Analyse ergab, dass dieser noch für die neue Funktionalität adaptiert werden kann. Der existierende PID-Kaskadenregler wurde, basierend aufs Konzept eines hybriden Position/Kraft-Reglers, adaptiert und verbessert, um einen Kraft-Sollwert zu akzeptieren. Zusätzlich wurden mehrere Verbesserungen in der Behandlung der gemessenen Regelgröße, die Kraft, unternommen. Diese inkludierten Verbesserungen zur Logik der Nullpunktmessung und die Inklusion eines nicht-linearen Median-Filters, um sich an das bestehende Design des Massageroboters anzupassen.

Table of contents

Foreword	3
Abstract	4
Kurzfassung	5
Table of contents.....	6
1 Introduction.....	7
1.1 Massage techniques, a short summary	8
2 Description of the system.....	9
2.1 Mechanical and electrical	9
2.2 Software and hardware.....	12
2.2.1 Human-Machine Interface (HMI)	12
2.2.2 Microprocessors	18
3 Scope and approach.....	21
3.1 Medical requirements	21
3.2 Feasibility analysis.....	23
3.2.1 Benchmarking	25
3.3 Approach to the solution.....	26
4 Implementation of the solution	28
4.1 Microprocessor adjustments	28
4.1.1 Assessment of the force controller	28
4.1.2 Process variable improvements.....	33
4.1.3 Changes to the force control sub-routine	34
4.1.4 Microprocessor limitations and solutions	39
4.2 Main program.....	40
4.2.1 Massage path file format	41
4.3 Tablet interface.....	41
5 Conclusions.....	45
6 Appendices	47
Appendix A: How to execute a massage.....	47
Appendix B: Location of project files and backups.....	48
Appendix C: How to directly flash a Delfino without CCS.....	49
Appendix D: Wiring diagrams	51
Appendix E: Program flowcharts.....	59
References	61
List of figures	63
List of tables	64
List of abbreviations	65

1 Introduction

The practice of massage is one of the few branches of traditional medicine¹ that stood both the test of time and the scientific rigour of modern medicine. With its first recorded uses dating back millennia and its practice having started simultaneously all over the planet, massages are very popular and ingrained into the customs of many cultures. Massage now provide many proven benefits and serve as effective treatments for many ailments as well as being considerably effective as a preemptive treatment for several health conditions [1] [2].

Many of these benefits, though, require regular treatments to be effective. This regular practice is very time consuming and due to this reason often neglected in hospital care. As the majority of developed nations in the world now struggle with a restricted budget for their health departments, lack of qualified personnel and a continuously aging population, there is a need to introduce automatization to the practice of massage to be able to introduce its benefits despite the organizational and economic hurdles.

Particularly, in modern society, Asian nations show a very strong interest in the use and research of massage techniques for clinical purposes. Almost exclusively, all performed research in the field of robotic massage therapy comes from either Japan, China or Korea. In these countries, the combination of a very strong electronics industry and a cultural inclination to value the effects of massages on health more highly [3], has led to east Asia leading in this field. Despite the early start, most of this research was done on niche, specific applications like: massage chairs [4], tapping robots [5], shoulder massage robots [6] and face massage robots [7]. There hasn't been yet a development to the scale of a fully automated hospital massage bed. Therefore, the need for such a product remains open.

The massage bed in this project is a prototype developed and constructed by Dr. Paul Finsterwalder and Dipl.-Ing. Klaus Bergkirchner for their respective PhD dissertations. This project continues their work with the prototype and goes deeper into the planning of massage paths and the application of customizable force profiles. Different massage techniques require different application forces and even different patients may have different preferences or pain tolerances that need to be considered. The goal of the continued development of this project through this master's thesis is to allow for more flexibility, customizability and ease of use when planning and deciding on treatments for the patients.

To achieve this goal two objectives had to be tackled. First, the development of a new interface that allowed for the creation of custom paths was necessary, opening for new possibilities diverging from the only two options previously available. This interface has to be user-friendly to allow operation and usage by non-technical users that have no knowledge of the inner workings of the system, and

¹ “*Traditional medicine refers to the knowledge, skills and practices based on the theories, beliefs and experiences indigenous to different cultures, used in the maintenance of health and in the prevention, diagnosis, improvement or treatment of physical and mental illness*” - World Health Organization.

additionally improve its functionality and options. This was accomplished by having a pressure sensitive table where the user can draw the desired path directly on an image of the scanned patient while at the same time storing the pressure values to simultaneously generate a force profile for the massage. Second, the structure of the system had to be adapted to process, transmit, and control the newly acquired force values for the massage. This had to be done while avoiding disrupting the operation of the existing functions of the system.

1.1 Massage techniques, a short summary

To properly understand the massages mentioned in this document one must know what the different massage techniques represent. The most relevant techniques for this machine are effleurage and kneading, as they are the ones that the machine can currently perform with its interchangeable massage tools (see chapter 2.1).

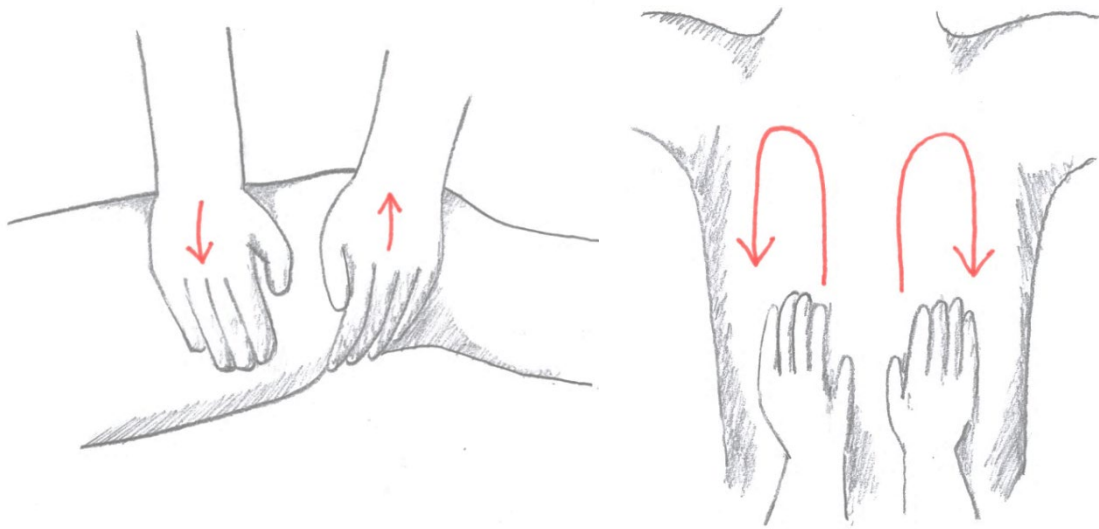


Figure 1: Massage movements for kneading (left) and effleurage (right)

Effleurage

Effleurage is a soft stroking massage where the hands glide or slide over the skin in long circular movements. It can be performed with the bottom of the fingertips and the complete palm with a light or medium pressure. It is normally utilized to soothe the patient and prepare for a firmer massage like kneading or to finalize the therapy session. This massage affects mostly the superficial tissue. It is most effective at moving blood through superficial veins and out of capillary beds. It has the additional effect of stimulating lymphatic return [8].

Kneading

Kneading (also called compressive petrissage) is a massage that applies pressure to compress the underlying muscles, separating them from the underlying structure. It is performed by using both hands on opposite sides of the muscle and pressing together to push the superficial tissue upwards. Its main purpose is to loosen and soften the tissue to allow for better local blood flow but if applied more energetically it can be used to tone the muscles [8].

2 Description of the system

The massage bed prototype is a system that encompasses many engineering disciplines like informatics, micro-electronics, electrical and mechanical design; together to create a complex mechatronic device. The bed has a massage carriage, also called arm, moved by an inverted portal robot that can use two different massage tools to execute two different types of massages: effleurage and kneading. From the carriage up to the massage tool touching the surface; the bed has six independent degrees of freedom. The original scope of the project started by the previous students projected a second massage carriage that could move independently of the first one, but this second one, was never implemented [1].

This chapter shall serve as an overarching documentation of all the work done by the different students in this project; concise but complete. It shall provide the basic knowledge of the inner workings of the system to understand the work done in the project and can also serve as an informed starting point for future students that may work in this project.

In the following section I'll list the components of the system, explain what their purpose is and how they interact with each other. An additional list of all the components can be found in Table 13 in Appendix D: Wiring diagrams.

2.1 Mechanical and electrical

The six degrees of freedom of the bed are each controlled independently by a dedicated external motor controller. The six axes are often named A1 to A6 in the hardware, documentation and software. The first three are the translation of the carriage and the last three are the rotation of the massage tool, as seen in Figure 2. The two coordinates θ_4 and θ_5 determine the rotation of the massage tool in spherical coordinates while θ_6 describes the orientation of the massage tool's head.

Axis	Coordinate ²	Description
A1	z (d1)	From feet to head of patient
A2	y (d2)	From right to left across the patient
A3	x (d3)	Vertical movement of the carriage
A4	θ_4	Polar angle of the tool
A5	θ_5	Azimuth angle of the tool
A6	θ_6	Rotation of the head of the tool

Table 1: Axes of the massage bed

² x, y and z describe the point where the massage tool makes contact. d1, d2 and d3 are calculated with an inverse geometric model and describe the point where the massage tool pivots. Regardless, both are coordinates in the cartesian domain.

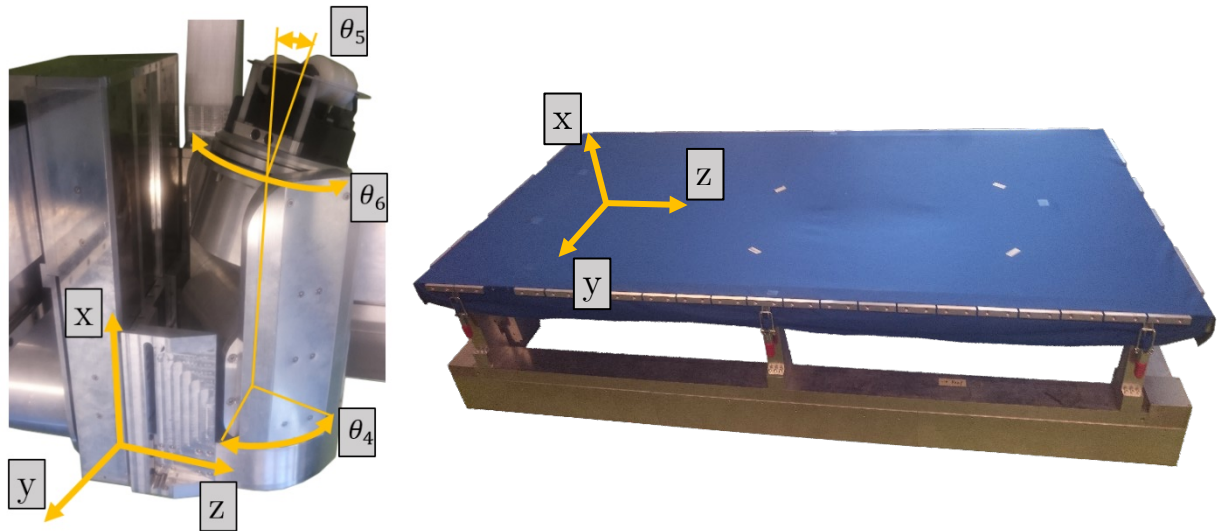


Figure 2: Movement axes depicted over massage carriage (left) and bed (right)

Each of the motors for the main axes has its own dedicated controller unit. These are external to the bed and provide the power for the motor directly. In each of these units there is a microprocessor of model TMS320F28335 by Texas Instruments (further called by its brand name: Delfino) that takes care of the control algorithm, calculating the power delivery and communicating with the other components of the system. Additionally, each motor has a rotary magnetic encoder on a small circuit board placed at the back end of each motor that is used to determine the motor's position. These values (along with force values measured by a strain gauge located inside the massage tool, for the A3-controller) are transmitted by an additional ARM microprocessor (STM32F030K by STMicroelectronics) in a dedicated communication line directly to the respective Delfino microprocessor.

All controllers communicate with each other over a custom fieldbus³ based on the RS-485 standard. The bus is commanded from an additional unit that then communicates with the computer. This unit contains three Delfino microprocessors, a main bus-controller and two so-called inter-bus transceivers that divide the system into two sub-buses. The controllers A1 to A6 are all inside a sub-bus (MB1) belonging to the first massage arm. The second sub-bus (MB2) communicates with the massage tool to receive the data from the performed scans (see chapter 0).

The two massage tools are named MK1 and MK2 (MK stands for "Massage-Kopf" meaning massage head). MK2 is used for effleurage and it has all its rollers oriented in one direction (Figure 3). The orientation of these is then constantly adjusted to always be parallel to the massage path. MK1 is used only for kneading as the rollers are oriented radially and the head applies a constant rotation controlled by θ_6 . Additionally, the MK1 tool moves the pressure point of the force, in circles, with a swashplate that simulates the technique used by real masseurs. The swashplate is controlled by an additional set of three motors that work together to reach the desired angle as seen in Figure 4.

³ A fieldbus (or bus for short) is a system that allows many electronic devices to communicate with each other over a single common connection by taking turns to send messages.

These motors use motor controllers of model DRV8808 by Texas Instruments and are connected to an MSP430F5510 microprocessor, also by Texas Instruments, that takes care of the calculations and communication with the bus. The two massage tools can be easily swapped in a plug-and-play fashion without fasteners or cable connections.

Power is provided to the controllers over a common line (PW1). They then deliver it to the motors over dedicated AC triphasic lines. The massage tools have a dedicated power supply line (PW2). The power supply must be set at a constant voltage of 42 V with current limits of ~10.25 A and ~0.75 A for PW1 and PW2 respectively. When turned on and in stand-by, the units drew 0.92 A and 0.08 A from PW1 and PW2 respectively. Any significant deviation from these values might indicate an electrical malfunction inside the system.

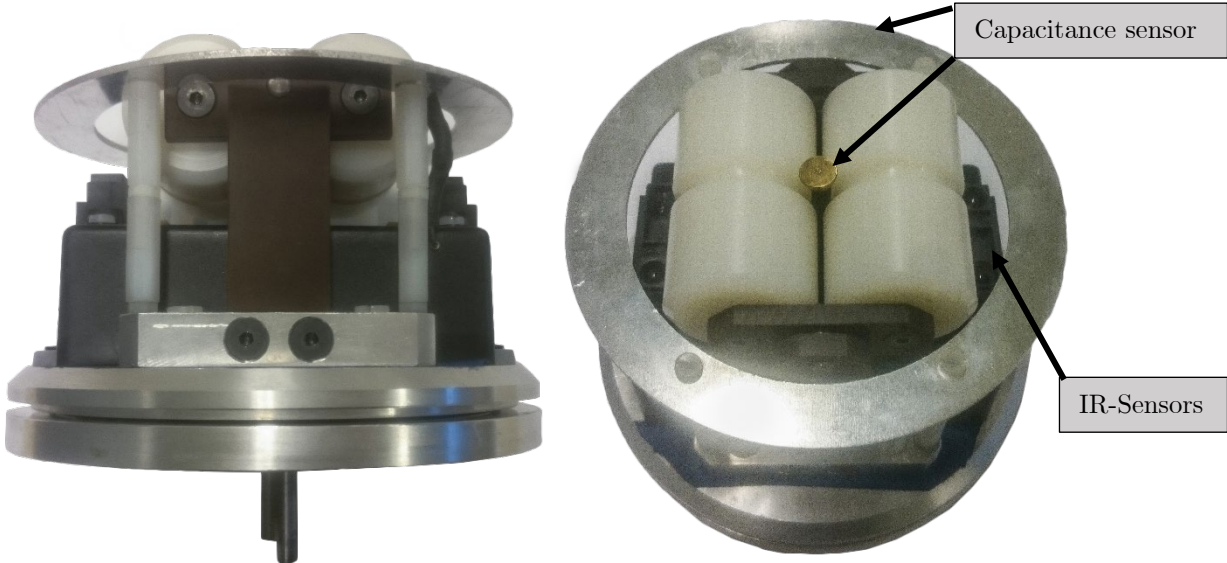


Figure 3: Massager tool MK2 with capacitance and IR distance sensors

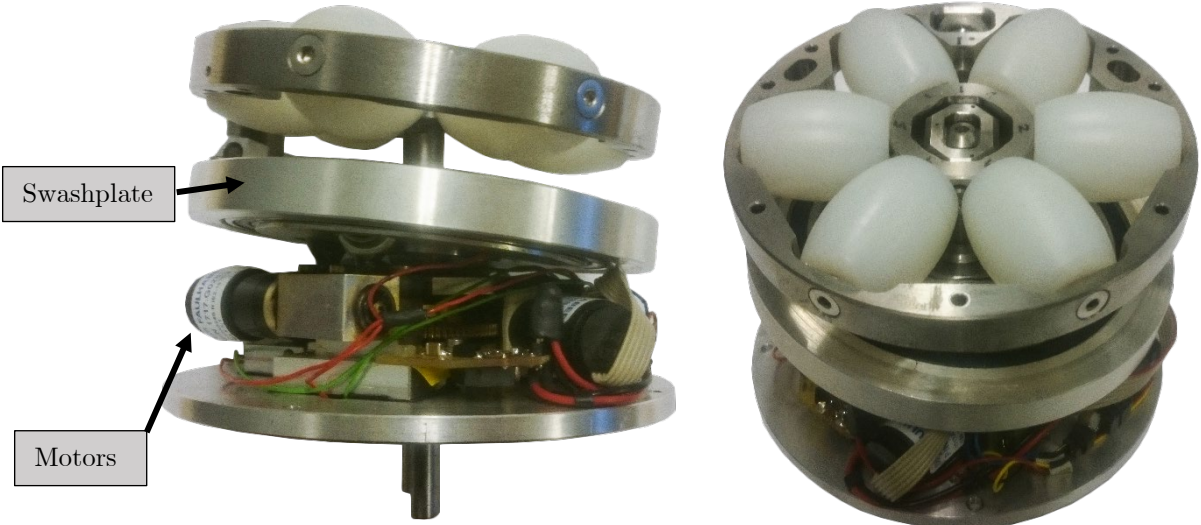
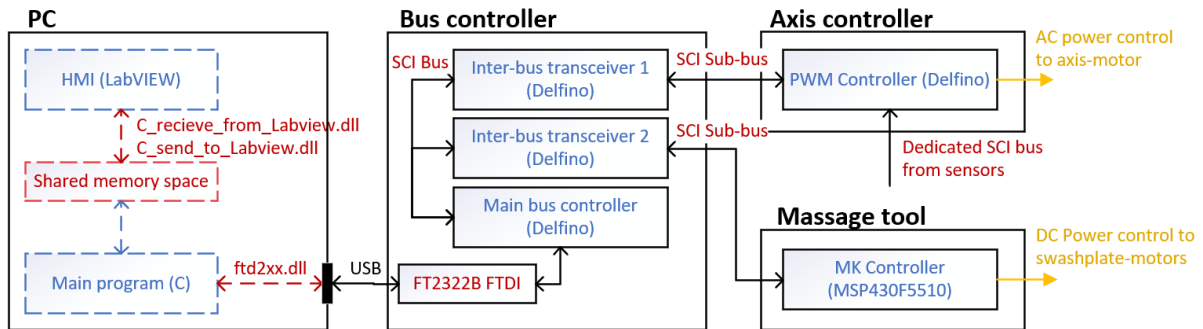


Figure 4: Massager tool MK1 with swashplate and motors

2.2 Software and hardware

The software package that drives the message bed and all its functionality is divided into three layers with different protocols to communicate between them. Figure 5 provides an overall view of all components and connections. The main parts of the software system are: the Human-Machine Interface (HMI for short), the main PC-program and the Delfino microprocessors; will be described in further detail in this chapter.



Legend:

Dashed line: Software element

Solid line: Hardware element

Blue text: Program

Red text: Communication protocol or intermediary

Yellow: Power control

Figure 5: Diagram of software and hardware connections

2.2.1 Human-Machine Interface (HMI)

At the top is the HMI. This is supposed to be the only point of interaction between the final users and the whole system. From it, all system functions can be accessed, and many features and parameters can be controlled using an actual user interface. This part of the software is programmed in LabVIEW (version 2011).

It is composed of two main interfaces plus a third one that was designed during this master's thesis:

- Main HMI
Source file: "HMI\HMI.vi"
- Amplifier interface
Source file: "HMI\Labview\Verstaerker_Interface1.vi"
- + Message path creation interface
Source file: "HMI\Message path creation interface\Message path creation interface.vi"

The main interface (Figure 6) provides the controls to initiate all the specific functions of the bed like scans, messages or zeroing all axes. It also displays the status of the components attached to the bus and if they are reporting any specific errors. Specific commands can be sent to a desired component in the bus with the "Send Message" function. "Sub_BUS" specifies the Sub-bus for either the axes' controller or the currently installed message tool. "Master_address" specifies which

specific microprocessor receives the command. Appendix A: How to execute a message, explains in further detail how to use this interface to execute a message from start to finish including scanning the patient.

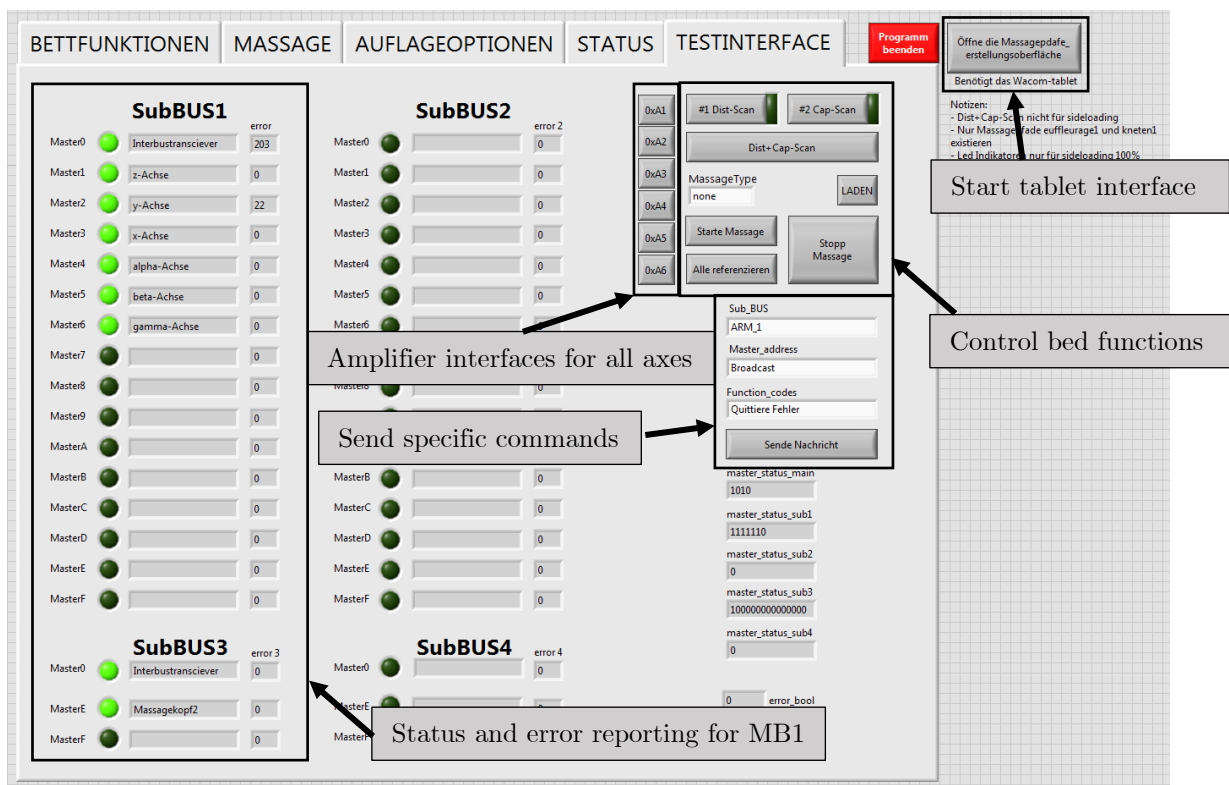


Figure 6: Functions of the HMI test interface

The amplifier interface (Figure 7) allows controlling the motor of a specific axis for debugging purposes. The control algorithm can be directly chosen at “Stromsollwertfunktion” (current setpoint function). This decides which value is used as a setpoint for the control algorithm. The three options are: position, rotational speed or electrical current and can be adjusted with the vertical bars. Using the rotational speed is the easiest way to manually move the axes with exception of the A3 controller (the vertical movement) that needs position control to stay at a desired position. For the A3 controller, the maximum force can also be set trough this interface. This value doesn't have any effect on the other five controllers.

At the bottom of the interface one can see the “program_version” that updates if “Status aktualisieren” (Update status) is pressed. Each controller has the program version hardcoded into their code and can only be changed by flashing a new version of the program with a different version number. The version number of the controller code before starting this thesis was 120 and it was then changed to 130 for the updated program installed for the A3 controller. At last, one can see the ten simulated LEDs for troubleshooting. These mirror the status of the actual LEDs inside the controller to be able to troubleshoot without opening the casing every time. More information to the significance of each one can be derived from reading the circuit diagrams.

This project adds a third interface as an integral part of the LabVIEW program: the “Message path creation interface”. This one will be explained in detail in chapter 4.3.

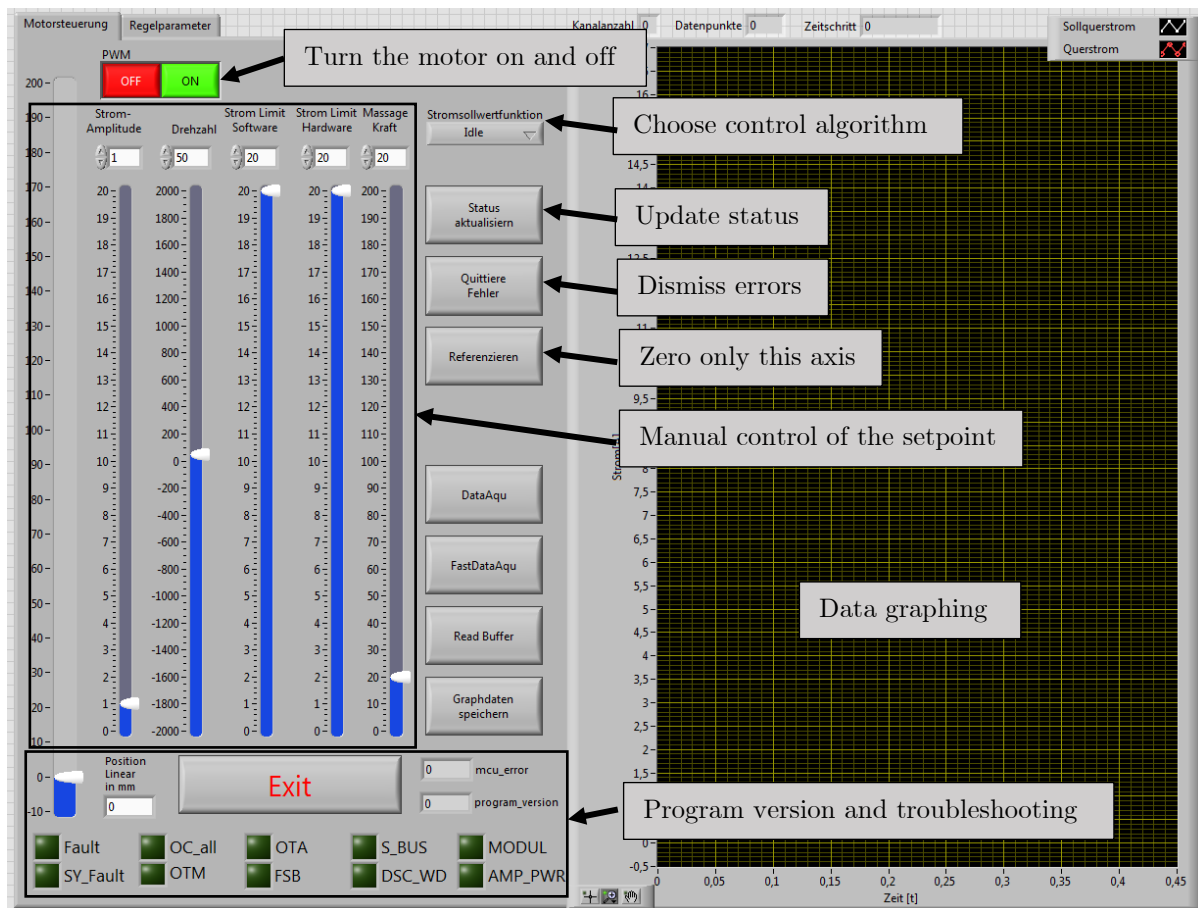


Figure 7: Functions of the amplifier interface

Main program

The main program is programmed in C language and executes automatically when the LabVIEW program starts. This program takes orders from the LabVIEW interface and does all the calculations necessary to create the message path which is then sent through the bus to the axes' controllers. The two main routines necessary to execute a message are the scans' analysis and the creation of the control values from a given message path. Its output is given inside a command-line window for debugging purposes.

Scans and analysis

Before starting any message, the program must be provided a scan result to adapt the path to the body of the patient. The bed performs two scans using the MK2 tool that has the scanning sensors. The first scan is performed with infrared sensors that measure the distance to the massage surface. This measures the descent of the massage surface due to the patient's weight. These values are then used to calculate the orientation of the massage tool to keep it perpendicular to the mattress at all times. The second scan uses a touch-less capacitance sensor that measures the capacitance of the object above; in this case, the body of the patient. The sensors can be seen in Figure 3. This creates an image of the body of the patient. The image is then analyzed to determine a set of body markers

that allow adjusting the massage path to patients of different sizes and body types. The body markers will be described in further detail in chapter 4.2.1.

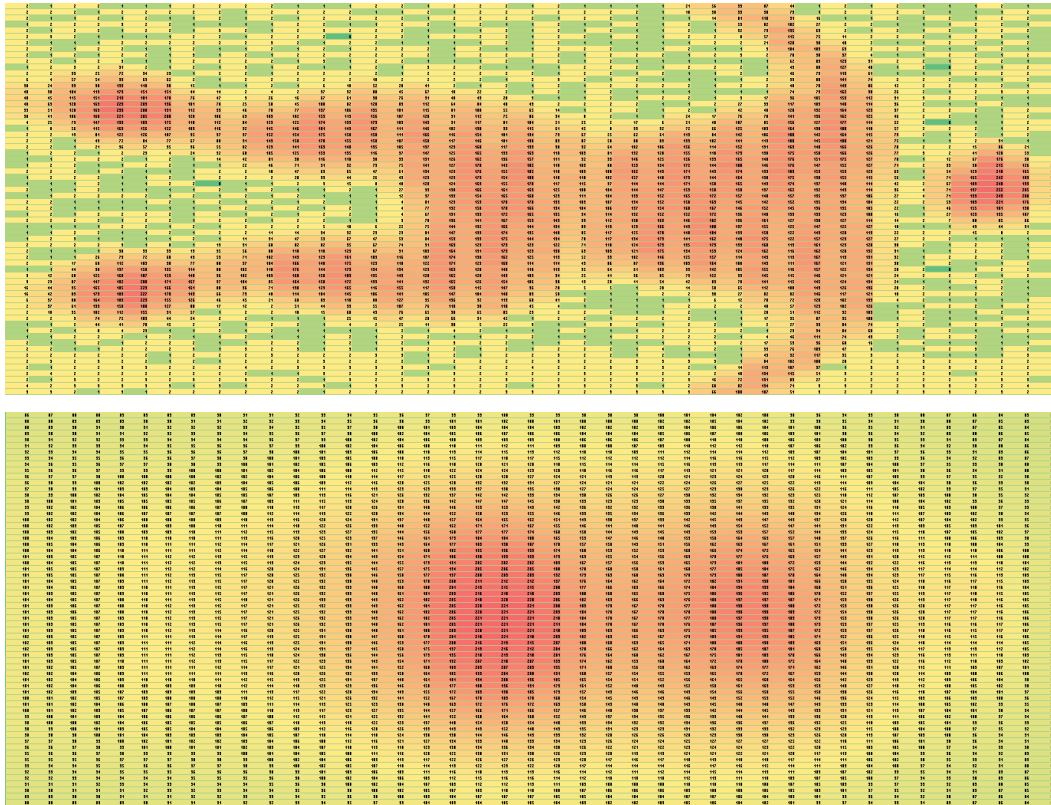



Figure 8: Capacitance scan (top) and distance scan (bottom)

When executing a massage for the first time on a patient, the scans must be performed from zero, but these can be saved for following massages. As long as the patient always places their torso in the same position, the massage will be executed in approximately the same way. Skipping the lengthy scan procedure saves valuable time of healthcare professionals and avoids inconveniencing the patients too often. The placement of the patient can be assisted by marking the massage surface with a specified position for the head and making sure that the rest of the body is parallel to the sides of the bed. The position of the arms or legs does not affect the adjustment of the massage paths in a substantial way so variations in their position can be disregarded. It is regardless recommended to keep the arms away from the body to help the body-recognition algorithm.

The scan settings are controlled from the main program’s header file: “main_program.h”. To perform the initial scan of a patient (or in the opposite way, load existing scans) the values must be set as Table 2 shows. These values are used by the C pre-compiler. This means that a change in these values requires a new compilation of the program. This can be quickly done by opening the

main program’s project file “bed_main.cbp” with “Code::Blocks”⁴ and clicking the gear icon , while the program is not running.

Setting	To perform scan	To load existing scans
“LOAD_C_SCAN_TESTFILE”	0	1
“LOAD_X_SCAN_TESTFILE”	0	1
“PERFORM_C_SCAN”	1	0

Table 2: Scans’ pre-compiler settings

The performed scans are saved every time in the following temporary files:

- Distance scan: “C:\temp\sideload_dist_mes.txt”
- Capacitance scan: “C:\temp\sideload_cap_mes.txt”
- Additionally: “C:\temp\cap_mes_temp_storage.txt”
(Used by the “Message path creation interface”, contains the capacitance scan and body markers)

To use these scans for future messages the files must be copied to another location to prevent overwriting. Existing scans can be loaded by placing them in the same folder where the .exe-file of the main program is located (“HMI\Main_Program\bin\Debug”). The files must be named as follows:

- Distance scan: “sideload_dist_mes.txt”
- Capacitance scan: “sideload_cap_mes.txt”

Loading message paths

The program can either: load one of the two pre-calculated paths for kneading (only for MK1) and effleurage (only for MK2) or load custom message paths; created with the message path creation interface (described in chapter 4.3) or by any other means as long as they respect the path format. These paths can also include a third column with information about the message force or not.

1. Effleurage default message path location:
“HMI\Main_Program\bin\Debug\effleurage\message_path1arm_single.txt”
2. Kneading default message path location:
“HMI\Main_Program\bin\Debug\kneading\message_path1arm_single.txt”
3. Custom message path location:
“C:\temp\custom_path.txt”

The appropriate settings for this can be found in the main program’s header file: “main_program.h”. They can be changed in the same way mentioned in the last section. If “USE_CUSTOM_PATH”

⁴ “Code::Blocks” is the C programming environment used for this project. Technically any other environment with a C compiler would work but the project has been already set up for this specific program.

is set to 1 the program will load the custom message instead of the default one. If “USE_FORCE_CONTROL” is set to 1 the x-axis will use “HEIGHT_TARGET” as a constant value for d3 (vertical coordinate) while controlling the force with “MAX_FORCE” as a setpoint. Additionally, if “USE_PATH_WITH_FORCE” is set to 1 the program will load and use the force data of the file and send it to the controller to use instead of the constant “MAX_FORCE” value.

The y and z coordinates specified in the message path represent a point on the flat message surface, but these aren't the values that are sent to the controllers. First, these points are projected down to the current descent of the message surface and so the x positions are calculated. The coordinates of the axes are then calculated by an inverse geometric model and in the end the six values for the axes are obtained: d1, d2, d3 and θ_4 , θ_5 , θ_6 (see Table 1). These values are multiplied by ten before being sent and used in the controllers.

Communication

The main program communicates with the controllers over the bus with a telegram system. The data or command is packaged into telegrams of a defined length and these are sent over the bus to be unpacked only by the correct recipient. There are four types of telegrams [1, p. 88] but two are the most important as they transfer the commands and data. They are divided into sections each one byte long that serve specific purposes and have the following structure:

AD	SA	TF1	FC	FCS				Command without data
AD	SA	TF2	LE	LEr	FC	PDU	FCS	Command with data

The individual sections are:

- AD Recipient address
- SA Sender address
- TFx Indicates a command: x = 1 without data / 2 with data
- LE Length of the data packet (PDU)
- LEr Length of the data packet again
- FC Function code
- PDU Data packet (LE bytes long)
- FCS Checksum

Table 3: Telegrams' structure

The maximum length of a data packet is limited and a whole message path doesn't fit inside a single packet so the arrays of coordinates for each controller are divided into 200 bytes packets and sent with a special routine called “SendPathMultiple()” that sends a sequence of telegrams to all controllers. The routines created for sending information over the bus are very complex and any changes done to them require a very thorough analysis. Specially in the case of routines that transfer axis coordinates because these not only transfer data for message paths but also for general movements of the message carriage and for scan paths.

2.2.2 Microprocessors

There are three main types of processing units in the system. The main bus controllers, the inter-bus transceiver and the PWM controllers. For this project the PWM controller is the only one that will be modified and looked into detail. A brief overview will be now given about the tasks of the other two.

Fieldbus controllers

The main fieldbus controller takes the role of intermediary between the PC and the SCI bus as the main program can't directly send electrical signals over the bus. It receives the telegrams in their respective protocols (USB from the PC and SCI from the main bus) from both directions and sends them their way. Inside the same enclosure and connected to the main bus are the inter-bus transceivers, MB1 and MB2. These serve as a filter for the telegrams to direct them to the correct sub bus corresponding to either the message carriage or the message tool [1, p. 85]. If a package or command is destined at only one of the two, the whole telegram is then packed as data inside a PDU destined to the corresponding inter-bus transceiver. The inter-bus transceiver will then unpack the telegram and send the data as an individual telegram to the desired controller through the sub bus.

PWM controller

The PWM controller works in conjunction with a second circuit board that has the hardware to provide the triphasic AC currents that directly power the motors. The controller receives the desired coordinates over telegrams and then executes a routine to ultimately convert them into the necessary current values.

The control program is structured with many different interrupt service routines (often simply called interrupts or ISRs) instead of a main loop that executes every cycle. These routines are executed only when they receive a specific trigger from the microprocessor architecture. The triggers for the ISRs are handled by dedicated silicon inside the microprocessors [9, p. 52]. This prevents unnecessary code from running constantly and frees up resources to run the necessary routines on a just-in-time basis.

Each controller must handle communication with two buses at the same time. Doing this with interrupts makes it easier to do asynchronously. The main ISRs of the program are called:

- Sub bus RX: "Scirx_isr()"
Sub bus TX: "Scitx_isr()"
- DMA buffers for telegrams: "DMA2_isr()"
"DMA3_isr()"
- Sensor bus (only RX): "SB_isr()"
- PWM controller: "epwm_isr()"

The receive and transmission of telegrams is handled by the sub bus RX and TX interrupts on a FIFO⁵ basis, respectively, but the actual processing of the telegrams is performed by the “RX_Delfino()” and “TX_Delfino()” sub routines. “RX_Delfino()” opens the received telegrams and moves their information to a dedicated buffer called “command_list[]”, through DMA⁶. This way all the telegrams can be received and processed quickly and be executed later, asynchronously. After being stored they are read by the “Command_execution()” sub routine that takes care of executing the specific actions corresponding to the specific command received. At this point the program diverges into the different functions.

“SB_isr()” takes care of receiving and processing the information sent from the sensor of each axis. This includes the data from the rotary encoders and the values from the strain gauges. It also executes the calculations to convert the values of the strain gauge to actual force values.

The “epwm_isr()” interrupt does the actual controlling of the motor. This is done by executing the “Regler()” sub routine. This sub routine can do the controlling with several different algorithms if told so by the amplifier interface of LabVIEW but by default, the one algorithm used is the one called “PATHCONTROL”. Inside this algorithm a series of functions are executed to perform the actual cascade controlling⁷:

1. “Control_value_loader()”:
Loads coordinates as setpoint and calculates the necessary speed between points.
2. “Trajektorienplaner()”:
Converts cartesian coordinates to the desired rotational motor position.
3. “Pos_f()”:
Control the motor speed using a P controller and the rotational position as a setpoint.
4. “Drehzal_f_dq()”:
Switching PI controller for the current values using either force or speed as a setpoint.
5. “interpolierer()”:
Interpolate the calculated current value between old and new values.
6. “Stromregler_dq()”:
PI controller for the triphasic motor-driving voltages.

⁵ FIFO, first in first out, means that all telegrams are processed in a chronological order.

⁶ DMA, direct memory access, is a hardware feature of the microprocessor that allows to move data inside the RAM to another location without utilizing the main processor’s pipeline and cache. This allows for much better performance [9].

⁷ Cascade controlling is a method of control typically used when controlling electric motors where the output is wrapped in several layers of control with the setpoint on the outmost layer. In this case there are 3 layers. The innermost layer, the output, is the desired current. The setpoint for each layer is calculated by the following layer. The second layer is the speed of the coordinate and in the outmost layer is the coordinate itself.

While all the controllers are set-up as PID controllers, all of them were originally adjusted to have no differential term ($K_d = 0$). The current stand of the controllers will be explained with more detail in chapter 4.1.3.

Disclaimer: Regrettably, as the work documenting these functions hasn't yet been completed by the original author [10]; the intent, exact inner workings and transfer functions of this controller are unknown. When writing this information great care was given to being as accurate as possible but these are deductions and conjectures made from solely reading the source code and might differ from an official description published by the developer in the future.

3 Scope and approach

The main objective of the project was to upgrade the functionality of the massage bed to be able to execute massage routines with a variable and customizable force. The provided code of the project developed by the previous students worked by using the controllers to control the height of the massage carriage, adjusting it to the descent of the massage surface due to the patient's weight. This applied a force only as a side-effect of shifting the desired vertical position upwards by 30 mm and therefore pressing against the patient. The obtained force could not be set to any value and was only controlled by an upper limit, for security reasons. This functionality was to be improved to allow for massage paths with controlled and varying massage forces that are relevant to the specific treatment of each patient.

To achieve this, the code of the different programs needed to be adjusted and new functionalities included. This isn't a problem for the program running on the PC as it has plenty of resources. But on the other hand, any changes made to the code running in the microprocessor might be critical to the program's execution. It had already been mentioned in the documentation that resources of the Delfino microprocessor might not be sufficient for implementing this functionality [1, p. 107] but it was never made clear on what metric this was determined. Therefore, a feasibility test was necessary to decide if the microprocessor had to be replaced with a more capable one or if the code could be changed in a way that allowed creating this new functionality within the limited resources of the system.

3.1 Medical requirements

As the medical aspects of the theory behind massage therapy and its benefits have been already researched in detail by Dr. Finsterwalder in his dissertation [1] this thesis will focus on providing information regarding specific forces of massage therapy and treatments. This is a subject where not much research has been performed across the world. This is due to the fact that massage therapy is a practice that is heavily based on the feel, expertise and intuition of the practitioner. The most tacit knowledge of a professional therapist is hard to describe in words and even harder to transcribe into numbers, as in the case of forces or velocities. These values would serve no purpose to masseurs when learning new techniques as they have no information about the applied forces or speeds they are applying besides their own developed sensation of touch. A second factor would be the fact that introducing a force sensor between the masseur and the patient interferes with their expected feedback, and with the ability of the practitioner to perform their techniques as usual. In textbooks used by the profession, the intensity of the massage is specified as "strong" or "soft", sometimes with more extensive descriptions of the expected effect on the skin of the patient. More information about the necessary direction, frequency or repetition is also provided. The following excerpt describes how an effleurage massage should be performed.

“Strokes are usually performed from the periphery to central or from distal to proximal and following the muscle fiber course. They can be executed in numerous variants: with one hand, with two hands

or with single fingers. The therapist makes contact with the whole hand (or both hands). The hands model themselves after the body surface.

Light strokes are applied with only so much pressure that the tissue fluids of the skin are displaced. The peripheral veins are streaked from distal to proximal. Because of their draining and tonus-lowering effects, stroking is a good start to the treatment. The frequency of draining strokes is low. A stroking-cycle of a limb occurs in three to five seconds. During this time, the hands are once guided from distal to proximal.

Strokes can also be used to stimulate or tone the muscles. For this purpose, a stronger pressure and a higher frequency are chosen. The dosage of the individual techniques is therefore dependent on the pressure intensity and the speed of execution. [2, p. 81]

A search across multiple databases of technical and medicinal research lead to other projects that implemented robots in the massage practice. These projects all focused on different types of massage techniques and as a whole give us a good idea of the spectrum of force needed to perform different types of massage. Regarding massage techniques used to massage the back, force measurement have been performed for relaxation massages, shoulder tapping, and individual-finger pushing actions.

When it comes to massages for relaxation of the back muscles, performed by robots, this is normally done in the massage-chair format. The technique used is basically an effleurage massage in the same way that the massage bed for this thesis performs it, with rollers that move with a constant pressure across the back of the patient. A massage chair might apply different levels of forces by pushing the back of the leaned-back patient with the rollers, lifting the bodyweight from its resting position. This is usually done with a simple mechanism that agnostically moves the roller closer or further without knowing what force values are being applied but these force values would vary depending on the patient’s bodyweight and body stiffness (coming from muscle and skin elasticity). For relaxation massages the comfort and personal sensation of the patients takes priority over predetermined routines. A study was performed using a massage like the one previously described that focused on achieving comfortable forces for a varied subset of patients [4]. The patients reported feeling comfortable with force values ranging from 4 N to 10 N with an average of 8.2 N. The data also shows that this subjective feeling of comfort does not directly correlate to bodyweight or body stiffness as shown in Table 4. This accentuates the importance of the feedback by the patient after the therapist has tested his massage routine; because premade routines with force profiles aren’t universal solutions that have the same results on all patients. The massage path creation interface should therefore include functionality to adjust the forces of the massage paths after their creation.

	Patients					
Bodyweight (kg)	55	53	44	90	59	56
Body stiffness (N/m)	376	430	276	192	159	190
Comfortable massage force (N)	9.2	9.8	4.0	8.4	9.0	8.8

Table 4: Relation between body-characteristics and massage comfort [4]

Research regarding pushing massage techniques with individual fingers such as pushing motions for shoulder massages or the Chinese traditional massage technique, Yi Zhi Chan (part of the Tuina

massage techniques), also recorded force values as well as force profiles for their movements. Both massage techniques utilized pulsating forces with relatively low forces with the Tuina forces being around and below 10 N and the shoulder pushing with values ranging from 4 N to 10 N depending on the finger of the hand [6] [11]. These kinds of pulsating techniques with small but very precise force values wouldn't currently be within the capabilities of our machine due to lack of measuring precision.

Some massage techniques do use considerably higher forces, than these soft massages. The measurements performed Dr. Finsterwalder in collaboration with professional therapists of the Kaiser Franz Joseph Hospital show force values reached by real massage techniques used for therapeutic purposes (not for relaxation). Friction and effleurage massages were performed over a custom-made 3-axes force measuring surface. The friction massage with fingertips pulsed between 50 N and a 100 N with about 1.5 Hz. The friction massage with palms pulsed between 50 N and 250 N with 0.5 Hz. The strokes of the effleurage massage sustained a high constant force of 300 N [1]. Lastly, information regarding forces used in tapping techniques was found that confirms the use of forces of 70 N for the percussive massage of the shoulder region [5]. This range of forces between 50 N and 200 N is more in order of what our machine was designed for.

3.2 Feasibility analysis

After a thorough analysis of the main program for the PC and the program for the different microprocessors it became clear that replacing the microprocessor with a faster, more capable one, could mean replacing the entire system. This is due to several reasons. In the best-case scenario, one would replace just the microprocessor, use the same code and reconnect all the peripherals; but in practice even the most compatible processor needs quite substantial work to adapt to the new system. The processors used inside the motor controllers are of the model TMS320F28335, belonging to the Delfino family of the C2000 microprocessors destined for low overhead controlling applications. The processors of the Delfino family offer partial code compatibility within the family assuming one does a series of code adjustments. The current processor has a frequency of 150 MHz, 68 KB RAM and 512 KB Flash memory⁸. If we wanted to replace it with a more capable one there are a couple of options (shown in Table 5). Right after the F2833X series came the C2834X series with double the processing power and a substantial amount of RAM, 516 KB, the highest of them all. But this series has a major drawback. It includes no on-chip Flash memory and no on-chip ADC. This CPU theoretically has the highest potential for the most complex code with its high RAM and expandable Flash but it's doubtful that this project will ever reach such complexity, even considering future development. Due to the large amount of work necessary to implement the Flash and ADC off-chip, this chip isn't recommended.

⁸ A KB, equally spelled kilobyte as the metric unit kB, is a common way to quantify memory capacity belonging to the JESD21-C standard. The correct technical unit and term for a KB are KiB and kibibyte, as defined in the IEC 80000 standard.

KB = KiB = 1024 Bytes ≠ kB = 1000 Bytes

A second option was the 28379X series that maintains the full on-chip design with quite substantial performance improvements. The F28379S is a single core while the F28379D is dual core. To take advantage of dual core in any meaningful way the whole program would need to be rewritten so this processor can't be given much consideration. That leaves the F2837S. Apart from the raw performance improvements, it offers an additional DAC that isn't present in the current microprocessor; and USB connectivity.

As a last consideration it should be mentioned that previous work in improving the processing power of the controller units has already been done by Mikael Gössl in his Thesis where he developed an improved controller prototype [12]. A circuit board has already been developed and the code been partially adapted for an ARM processor with better performance than most of the Delfino microprocessors, as seen in Table 5.

Processor	CPU	Frequency (MHz)	Flash (KB)	RAM (KB)	Channels			USB
					ADC	DAC	PWM	
F28335	C28x	150	512	68	16	0	12	No
C28346	C28x	300	0	516	0	0	18	No
F28379S	C28x	400	1024	164	24	3	24	Yes
F28379D	2x C28x	200	1024	204	24	3	24	Yes
Processor used in the improved controller prototype [12]:								
ADSP-CM408F	ARM	240	2000	384	24	0	12	Yes

Table 5: Considered processors' specifications

If one were to implement the F28379S in the system, the code would need a thorough analysis. Most peripherals are still supported but some have changed versions and eCAN as well as XINTF are not available anymore [13]. Additionally, the code will need adjustments in several aspects. *“GPIO mux numbers, options, and code will need to change. Clocking configurations, security, and general system control is a new architecture.”* -Texas Instruments Employee; as a response in their forums to an inquiry about this topic. The processor is still available in the same package, LQFP⁹ with 176 leads to the sides, but the pin allocation has changed and therefore the circuit boards would need to be redone.

Due to the lack of documentation for the design and intent of the motor controllers and their aggregates (like the additional microprocessor that takes care of sending data through the dedicated SCI sensor bus, Figure 5) the intended behavior of the system as well as the complex intercommunication of the different parts would have to be assumed entirely from the source code if any adjustments were to be made. Additionally, this code has only been very sparsely commented, leaving a lot to speculation. There is always the risk that, even after a complete

⁹ LQFP stands for Low profile Quad Flat Package. The F28379S actually has a HLQFP package that basically a thicker LQFP with better thermal capabilities. The H means thermally enhanced.

replacement of the processor, some non-foreseen or non-documented interaction between the bus participants prevents this complex system from working properly again.

Due to the introduction of a new processor being a high risk and very long task it was left as a last resort if other methods were determined as inappropriate or not sufficiently fast. The first task was, therefore, to benchmark the speed of the system to estimate its suitability for the planned adjustments.

3.2.1 Benchmarking

To determine the impact of any code changes done to the main controller code of the microprocessor first a baseline has to be determined for an average execution time of the controller routine. The internal values of variables in the processor can be read through Code Composer Studio (CCS for short) if the processor is connected to the PC through a JTAG emulator¹⁰. In this case though, to avoid any kind of unintended delays or slowing down of the processor, an alternative method was used that avoided the use of the JTAG emulator and the intervention of CCS. To measure the cycle time of the controller routine the state of one of the GPIO pins¹¹ of the processor was toggled from within the code every time the controller completes a cycle. This method utilizes a single line of code and it being so lightweight prevents affecting the value to be measured. The state of this pin is connected to the Test-Out port already incorporated into the initial design. A ribbon cable was used to make this port more easily available for testing when the case of the controller is closed. The changing digital value on the pin can then be measured with an oscilloscope (Model of the employed oscilloscope: Tektronix MSO 4034) which easily tells us the frequency. This value will be later compared to the values obtained after modifying the existing code to include the additional functionality.

The controller cycles aren't always of the same length. They may vary depending on the amount of code to be executed before starting the next cycle. This can be seen in Figure 9 where each flank represents a new start of the routine.

It was measured that the average cycle time of the "Regler()" routine was 19.73 μs with a standard deviation of 0.34 μs , a minimum of 7.67 μs and a maximum 34.64 μs .¹² This equates to a frequency of about 50 kHz. If one takes into consideration that

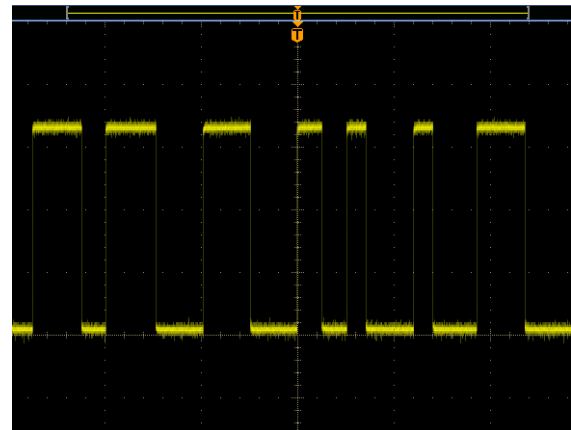


Figure 9: Varying cycle times when benchmarking routines of the microprocessor

¹⁰ Joint Test Action Group, also known as JTAG, is a standard for verifying and debugging circuit boards. A JTAG emulator can connect directly to the memory of the microprocessor to read and write even while the microprocessor is running.

¹¹ GPIO pins: General Purpose Input Output pins for digital signals.

¹² This average was obtained taking the averages of the cycle times already calculated by the oscilloscope and averaging them together. In the measurements T/DIV equaled 20 μs .

the massage paths only specify a coordinate for every 100 ms (the intermediate points are interpolated, see Table 12) that would mean that this routine is theoretically executed 5000 times between each coordinate in the path. This is already a quite high temporal resolution that could still decrease significantly without affecting the movement of the massage carriage or having the patient notice. Additionally, the rate of arrival of data from the strain gauge was measured with the method. The results showed that this data is delivered on average every 144.94 μs with a standard deviation of 5.58 μs , a minimum of 75.19 μs and a maximum 204.2 μs .¹³ This would mean that the force controller could even run 7.25 times slower and it would not make any difference, as the controller would not yet have a new value from the sensor to make any calculations. This data is again summarized in the following Table 6.

	Average (μs)	Std. dev. (μs)	Minimum (μs)	Maximum (μs)
Controller cycle time	19.74 (50.6 kHz)	0.34	7.67	34.64
Strain gauge data arrival rate	144.94 (6.9 kHz)	5.58	75.19	204.2

Table 6: Summary of data for the benchmarking

Considering this information, it was determined that it is possible to implement the force controller and additional functionality without slowing down the program in any remarkable way. Due to the high temporal resolution of the controller any slowing down would most certainly be imperceptible to the patient.

3.3 Approach to the solution

Being the objective of the project mainly to provide the functionality of force control, theoretically only one of the six controllers needs to have its code adjusted.¹⁴ The controller in focus is the A3 that controls the vertical position of the massage carriage. In the original program, in a normal execution, the controller follows the given x values and the applied force takes only a secondary role. There is a section of code dedicated to the control of the motor by force instead of position, but this section is only activated if a pre-set maximum force was exceeded. This force value acted practically only as a safety limit that cannot be surpassed but in no way as a desired force value as there is no minimum to the force. This leads to the massage tool sometimes moving while not even touching the massage surface, as there is no incentive to move upwards at that moment.

¹³ As mentioned in the previous note this average is an average of oscilloscope-calculated average values. In the measurements T/DIV equaled 100 μs .

¹⁴ In practice, all six controllers share the same code as they all do basically the same function; control the revolutions of their assigned motor to follow the received coordinate values. They differentiate inside the code using different subroutines depending on the controlled axis. Therefore, any changes made must consider what would happen if another controller were to execute the program.

Controlling the vertical position of the carriage and the applied force at the same time is physically impossible as the force is created by the body weight of the patient when the massage tool presses against it. The higher the force needed, the higher the tool must go. The maximum possible force that can be exerted on a patient is physically limited by the weight of the patient itself and the maximum height that can be reached by the massage tool. From these, the only factor under our control is the height of the tool and its maximum is the height of the massage surface without a load. This led to the realization that the height of the massage carriage doesn't really need to be controlled as it is only an intermediary to apply a force. Its value can be hardcoded into the program as the maximum and let the force be controlled instead, with the desired values sent from the PC. The only hard limitations to the value of the force would be security features¹⁵ and body weight, leaving a range of freedom to the healthcare professional. The detailed explanation of the implementation of this solution will be described in the next chapter.

¹⁵ The force is limited with current limiters, both digitally inside the program of the axes' controllers and electrically in the circuit board.

4 Implementation of the solution

This chapter describes the changes made to the programs of the system to accommodate for the force control functionality. It also talks about the new interface and program created to generate the custom message paths with force information, using a touch tablet and pressure sensitive pen by Wacom (Model: Intuos Pro).

4.1 Microprocessor adjustments

As mentioned earlier, the program of the axes controllers already includes code to implement force control. This sub-routine is in many ways fundamentally different and simpler than the sub-routines that control the position. The controller for the force was only implemented as a safeguard for the position controller not to exceed a certain force value. Regardless, after analyzing the code it was estimated that it may be sufficient to achieve the custom force profiles if tuned properly. Therefore, the first task was to assess the quality of this controller with simple tests, static loads and message paths.

Equally important as the control law of a controller is the quality of the measured process variable. In this case, the momentary force being applied by the tool. In this machine the force is measured by a strain gauge integrated into the message carriage. Analyzing the information received from the strain gauge can give us valuable information. The quality of the signal, depending on noise, electrical fluctuations or mechanical oscillations can tell us what amount of precision can be expected because no value can be precisely achieved if its baseline isn't at least as precise as the desired result. From the retrieval rate of the data we can determine what is the maximum meaningful refresh rate that the controller could have.

4.1.1 Assessment of the force controller

The strain gauge that measures the force is accompanied by a small processor that transfers the measured data over an SCI bus to the respective controller. We want to gain access to this value for the upcoming test. The data is transferred digitally and the Delfino processor has no DAC unit, so these values cannot be read by an oscilloscope. As an alternative to measuring them with hardware, when the computer is connected through a JTAG probe¹⁶ to the microprocessor, CCS can read values directly from the memory of the processor. The major limitation to this method is that the maximum poll rate is harshly limited by CCS. In optimal conditions with a low PC load it may reach 10 Hz [14]. The desired refresh interval can be set in Preferences>Code Composer Studio>Debug>continuous refresh interval, and its minimum is 100 ms, equaling a poll rate of 10 Hz. This means that only 0.145% of the values are recorded.¹⁷ When evaluating the CCS debug

¹⁶ The JTAG probe used for this Project was the Blackhawk USB200 JTAG Emulator (BH-USB-200).

¹⁷ Refresh rate of CCS/strain gauge data arrival rate = 10 Hz/6.9 kHz = 0.145% (see Table 6)

mode graphs, one must remember that the refresh rate inputted into the properties of the graph is just a scaling value for the time axis and in no way relates to the poll rate for the graph.

Several measurements were made to evaluate the quality of the provided force signal and the existing method of force control. These measurements were performed using a special mode of the system where the d3 values (vertical coordinates) are replaced by their maximum safe value of 160 mm before being sent. This assures that they will reach the maximum force they can exert on a given subject, as explained in chapter 3.3. To turn on this mode one must change the “USE_FORCE_CONTROL” setting (found in the main program’s header file, “main_program.h”) to 1 and recompile the program. Additionally, “USE_PATH_WITH_FORCE” (found in the same section) must be set to 0 as no force profiles will be loaded.

Static measurements:

To analyze the stability of the measurement of the force, a defined static load test would serve as a baseline to determine the precision and variability of these values. This test was done by directing the massage carriage hands-on with the amplifier interface (Figure 7) and placing a weight on it. The test position was in the center of the bed as to avoid additional load from the massage surface being restrained from the sides. The vertical position of the massage carriage was 150 mm. This places the massage tool at approximately the level of the massage surface without a load. The weight had a mass of 1.316 kg and the program was set to a limit of 50 N maximum. When measuring force values without using a massage path the force must be zeroed manually with a command from the HMI when on position. The force values as well as the unmodified strain gauge values were recorded.

On a first glance it can be observed that the values obtained aren’t very stable; not even when the arm isn’t under load. This could be caused by several different factors like electrical interference or due to the lack of precision of the sensor when measuring relatively low force values. The theoretically expected force of the weight would be 12.91 N. The average of the measured force under load was 11.41 N with a standard deviation of 2.2 N. When adjusting for the average under no load of -0.96 N it results on a measured force of 12.37 N. This is a discrepancy of 17.8%. The measured data is shown in Figure 10 (left). The standard deviations shown in Table 7 give us information about the stability of the values that would be used by the controller.

	Average with load	Std. dev. with load	Average no load	Std. dev. no load	Adjusted value ¹⁸	Maximum deviation (%)
Force (N)	11.41	2.2	-0.96	2.17	12.37	17.8%
DMS (-) ¹⁹	4041	68	4436	71.1	394	17.2%

Table 7: Summary of data for the static load measurements

¹⁸ The adjusted value is the average with load minus the average without load.

¹⁹ The strain gauge values will be referred from now on by their useful German abbreviation, DMS. They have no physical unit.

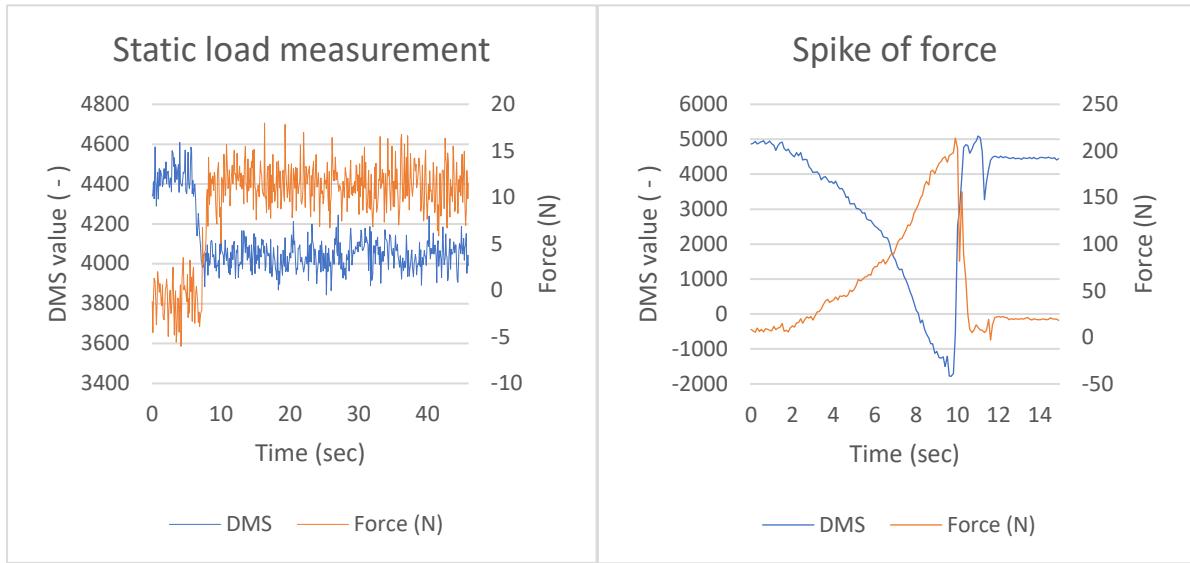


Figure 10: Recorded force and strain gauge (DMS) values for a static load (left) and manually induced current safety limit (right)

Additionally, the maximum force that the tool can reach before shutting the motor down due to the internal security limitations to current in the software had to be determined. By pressing down on the tool, it was measured that this value lies around 220 N as shown in Figure 10 (right). For the safety of the system, future massage paths including force values should not surpass 200 N.

Dynamic measurements:

These measurements were performed during the default effleurage massage path performed on a water pillow to simulate a patient and using body markers based on previous values generated by a scan of a real subject. The pillow is approximately placed in the position of the left side of the back of the patient and it is filled with water so that it weighs 4.7 kg. The test setup and markers are shown in the following Table 8.

Marker ²⁰	y (mm)	z (mm)
KS	296.66	1600.00
SS	308.33	1280.00
HS	325.00	880.00
SL	609.74	1292.56
SR	50.22	1269.24
HL	489.86	886.87
HR	190.12	874.39

Table 8: Dynamic measurements' test setup information for reproducibility

²⁰ The format and meaning of the markers are explained in chapter 4.2.1.

Several measurements were performed to study the performance of the integrated force controller. It was important to observe how much the values approached the specified force limit and what margin of error they had. This data shows negative force values previous to the massage beginning because its value is zeroed only right before it begins.

In the measurements with a limiting force value of 10 N it can be observed that the force is clearly cut at the 10 N range (Figure 11 left) but the actual average is 8.76 N with a standard deviation²¹ of 4.06 N and a median of 8.98 N. This means that these values fluctuate by about 46%, and it can be clearly seen that some values even become negative. When compared with the DMS values for the same section it can be seen that their maximum deviation is only 15.6%. This is due to the multiplicative conversion of the DMS values to force values, as seen in equation (1). Every deviation in the DMS from its expected adjusted value generates a deviation in the force of: $Force\ dev = DMS\ dev \cdot 0.030637$.²² Due to the order of magnitude of the DMS values being two times higher²³ the deviations of the force end up being around 3 times the deviations of the DMS. This relation changes for higher forces, tending to better stability, as can be seen in the last column of Table 9. This is a good sign as most massages require considerably higher forces than the ones measured here, but nonetheless any effort done to improve the stability of the DMS value would have 2 to 3 times the positive effect into the stability of the force, which is very important for the force controller.

$$- Adjusted\ DMS\ value \cdot 0.030637 = Force \quad (1)$$

$$Adjusted\ DMS\ value = Current\ DMS\ value - No\ load\ DMS\ value \quad (2)$$

Theoretically the water pillow can exert a maximum force of 46 N, but this wouldn't occur in a normal situation as part of the weight is carried by the tarp. Further measurements were made to see how the force controller behaves when not constrained by the force limit. The data measured for a 30 N and 60 N force limit shows that the force is not cut by the limit and that it oscillates around the average of approximately 16 N. To determine if the oscillation was also present in some form in the constrained force massage path of 10 N (meaning that the controller let it trough), an FFT (Fast Fourier Transformation) was performed on the data of both measurements. In Figure 11 (right) it can be seen that the relative intensity of the constant part of the spectrum is one order of magnitude higher than the rest of the spectrum. In the non-force-limited massage we can observe a ~10 Hz oscillation in the FFT and its relative intensity has the same order of magnitude as the constant part of the spectrum (Figure 12, right). All the data corresponding to the dynamic measurements is summarized in Table 9.

²¹ The standard deviation of the dynamic data was calculated as the average of a moving standard deviation with a window of 7 samples (700 ms). The approximate size of this window can be seen in Figure 13.

²² Value taken from the source code.

²³ Values around 4000 for the DMS and around 40 for the force, 100 times bigger.

	Force limit (N)	Average with load	Std. dev. with load	Median with load	Average no load	Adjusted value	Maximum deviation (%)
Force (N)	10	8.76	4.05	8.97	N/A	N/A	46.3%
DMS (-)	10	4456	112.5	N/A	5236	780	15.7%
Force (N)	30	15.66	3.42	15.59	N/A	N/A	21.8%
DMS (-)	30	4215	114.6	N/A	5227	1012	11.3%
Force (N)	60	16.3	3.31	16.8	N/A	N/A	20.3%
DMS (-)	60	4071	114.4	N/A	5085	1014	11.3%

Table 9: Summary of data for the dynamic measurements

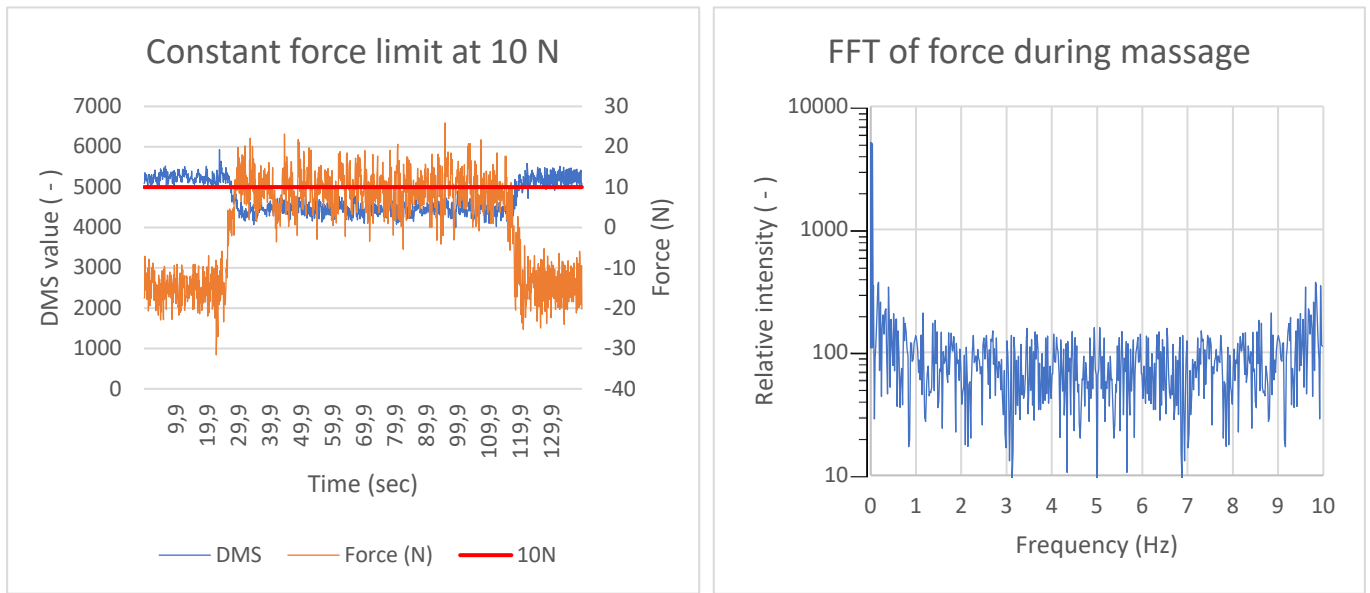


Figure 11: Recorded force and strain gauge (DMS) values for force limiting at 10 N (left) and FFT of the force values during the massage (right)

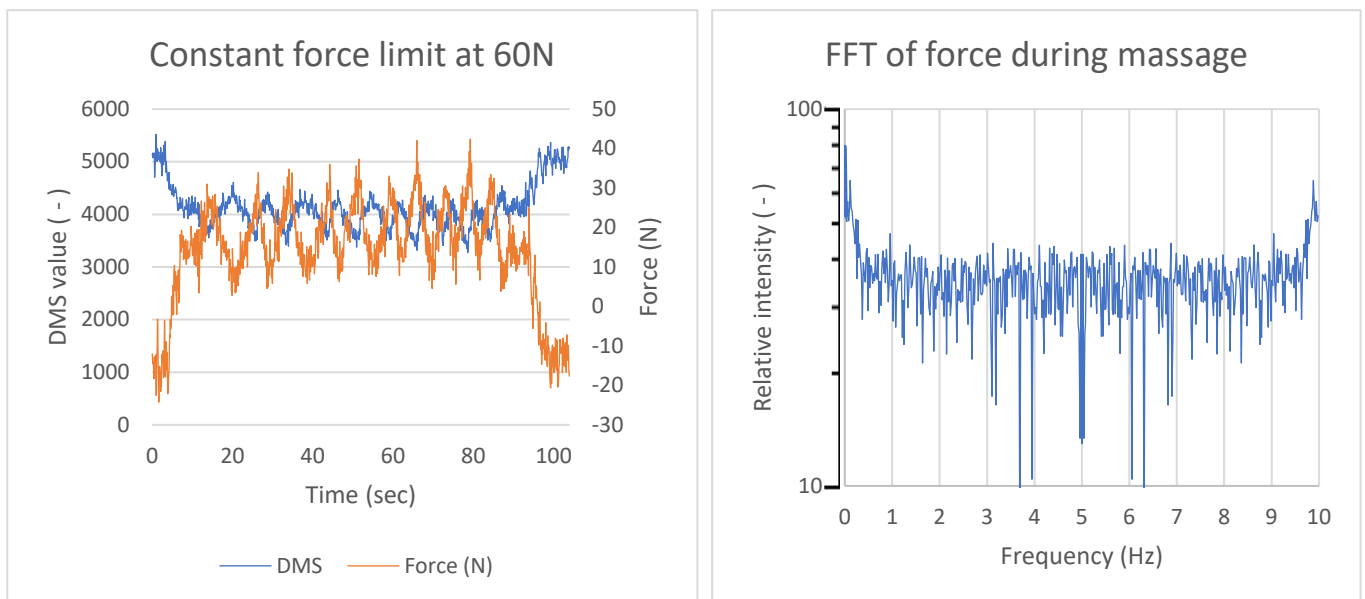


Figure 12: Recorded force and strain gauge (DMS) values for force limiting at 60 N (left) and FFT of the force values during the massage (right)

4.1.2 Process variable improvements

To improve the stability of the strain gauge values, these must be pre-processed with some kind of low-pass filter to remove as much noise as possible. As an alternative to using an analog low-pass filter on the electrically measured values, a programmatic nonlinear median filter like a moving average can be used. The filter was applied on the DMS value before converting it to force values. As this processor is quite limited in speed and RAM, special attention was placed on the datatypes used as well as on the data structure for the queue to store the incoming values. A so-called ring/circular buffer was used to store a predetermined amount of values that are then to be averaged. A ring buffer provides a speed of $O(1)$ in comparison to $O(n)$ for a traditional array when storing a new value and deleting the oldest one.²⁴ The moving average was tested with windows of 10 and 100 values. While the function itself is certainly taking longer to run, the cycle time measurements proved that this implementation did not affect the execution time in any measurable way. That means that the window for the average could be further increased if necessary, but this is limited on the other side by the memory usage that has already reached its limit. Alternatively, the window size can be artificially increased by using only one out of n values. This would practically increase the window size by a factor of n without utilizing more memory, but it would sacrifice temporal resolution.

The measurements summarized in Table 10 show a measurable and significant improvement in the stability of the reference value. When compared against the values obtained in the previous dynamic measurements (Table 9) the maximum deviation dropped from 11.3% to 6.2%, 5.4% and then 4.0% for windows with 10 samples, 100 samples and 100 samples with $\frac{1}{2}$ dropped values, respectively. Figure 13 shows an exemplary superposition of the unmodified DMS values over the ones averaged with a 100 samples window. It is clear that the averaged ones have less spikes and overall less fluctuation. From this point on, the 100 samples moving average was used in all measurements and tests. Additionally, all negative values for the force will be limited at zero to reduce signal noise as a negative force has no significance in this application.

Average window size	Dropped values	Average with load	Std. dev. with load	Average no load	Std. dev. no load	Adjusted value ²⁵	Maximum deviation (%)
10 samples	None	3897	72.7	5035	49.2	1138	6.2%
100 samples	None	3942	60.5	5063	33.4	1121	5.4%
100 samples	1/2	4064	52.4	5386	31.9	1322	4.0%

Table 10: Summary of measurements for the improvement of the stability of the reference value (all are DMS values without physical units)

²⁴ Big O notation gives information about the amount of calculations or instruction needed to process one element. $O(1)$ means one instruction per element while $O(n)$ means n instructions per element where n is the length of the array.

²⁵ The adjusted value is the average with load minus the average without load.

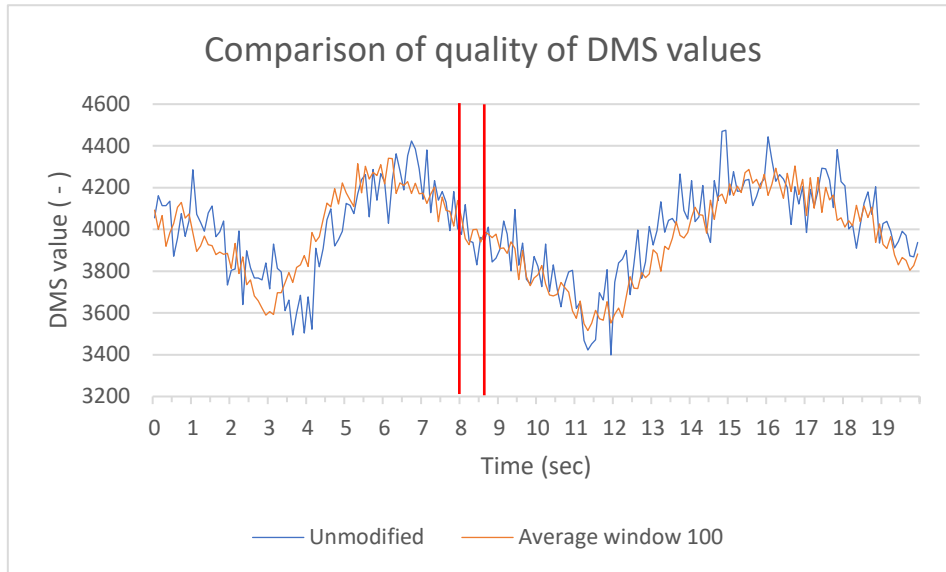


Figure 13: Superposition of the unmodified DMS values and the averaged ones (window for calculation of the moving standard deviation shown with vertical lines)

4.1.3 Changes to the force control sub-routine

The controller designed by Klaus Bergkirchner for the axes of the massage bed was based on a hybrid approach to force/position control. This controller switches between a traditional cascade controller, as commonly used by CNC axes, that controls the motor currents based on the desired position, and a second controller that also controls the motor current but based on a desired force (without cascade control). Both of these use PID control laws on each of their levels of control, except for the position having solely a pseudo-PD controller. Figure 14 and Figure 15 show the block diagrams for both controllers.

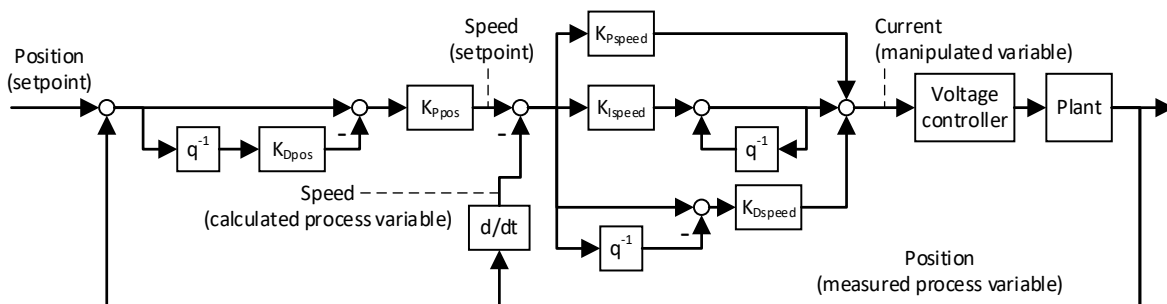


Figure 14: Block diagram for the position controller

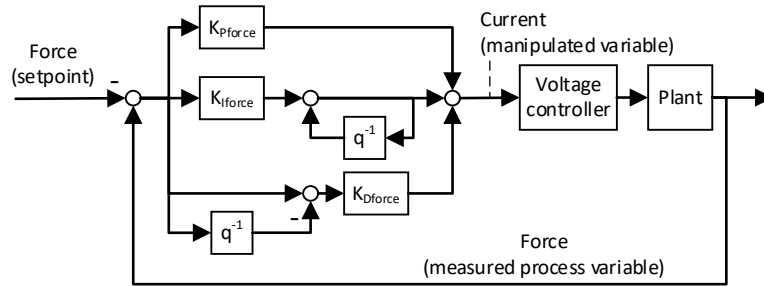


Figure 15: Block diagram for the force controller

On a hybrid (also called switching) controller the application switches between the two completely separate control laws depending on a specified condition. The first necessary improvement was to modify this switching logic so that less switching between the two control laws occurs during a massage. The objective was to give the force controller precedence, to be able to observe its dynamics independently of the speed controller and therefore tune them appropriately. The concept for the chosen switching logic is detailed in Flowchart 1 in Appendix E: Program flowcharts. With this new logic the force controller leaves behind its role as security feature and now controls the vertical movement during the complete massage. This whole logic is only active if the “USE_FORCE_CONTROL” parameter is set by the main program, using a telegram.

To detect the point when the massage starts and differentiate it from general movements for positioning or scanning, the main program sends a telegram signaling the start of the transmission of massage coordinates. From this point on, until the receipt of a second equal telegram signaling the end of the transmission of coordinates, all massage coordinates are bundled with this information inside a structure. If a coordinate is part of the path and the force exceeds the force setpoint multiplied by “FORCE_THRESHOLD”, the force controller activates. It will then only deactivate if the carriage moves up until the point it reaches the “HEIGHT_RESTRICTION”. After switching to the position controller, the value “HEIGHT_TARGET” is used as a setpoint for the position.²⁶ When a set force cannot be reached, the massage carriage will translate between these two heights. This behavior can be observed in Figure 16 (left).

Stabilization of the baseline controller dynamics

After implementing the filter on the DMS values, several dynamic measurements were performed, without any load on the tool, at a fixed height. These measurements showed that when moving the massage carriage up or down quickly, big force spikes and fluctuations are registered on the force sensor. This happens because of the presence of a considerable mass between the strain gauge and the point of contact of the massage. Whenever the carriage accelerates (as it was set to do so very quickly) the whole mass of the vertical axis exerts a force on the sensor. This acceleration force is

²⁶ These constants can be found and changed inside the “Regelung.h” header-file in the PWM_Regler program. Note: The measured process value for the vertical position that the height restriction is compared against, equals the x coordinate target + 12 mm. This difference must be accounted for when setting a height target.

quite problematic as it prevents us from measuring the actual force being applied. Additionally, the spikes can accidentally trigger the force controller at an undesired moment. Without changing the mechanical design one can only mitigate the problem but not completely solve it. The ideal solution would be to bring the measurement of the force closer to the contact point. This would minimize the accelerated mass and reduce the error. A possible location for the force measurement on a future prototype could be on the shaft of the massage tools themselves (see Figure 3 and Figure 4 in chapter 2.1). Additionally, the use of a commercial sensor instead of self-applied strain gauges would be recommended to obtain better values and minimize the development effort. The bottom of the shaft of the massage tool could be directly supported by a circular load cell to measure the applied force.

To mitigate the problem in the current machine it was chosen to adjust the settings of the controller to limit the acceleration of the system. These settings are sent to the controller every time the main program starts but can also be changed through LabVIEW using the amplifier interface. The initialization settings for all controllers can be found in “HMI\Main_Program\bin\Debug\ini”. The main tweaked setting was the maximum allowed acceleration that was reduced by a factor of a hundred. This value was probably originally set very high to utilize all the available acceleration as the force was not a concern in the original design. Additionally, the maximum speed was reduced by two thirds and finally the speed controller parameters were tweaked down to further improve the stability of the disturbance response by increasing its settling time. At this point a coefficient for the differential term of the PID controller was tested to further limit acceleration at the control level. Previously none of the controllers used a differential term.

To compare the effectivity of these settings at stabilizing the measurements, tests were performed with a reference massage using force control and a setpoint of 10 N but no load on the massage tool (Figure 16 left) at a fixed “HEIGHT_TARGET” of 130 mm. Using the original settings, the force measurement accidentally and repeatedly surpassed the setpoint of 10 N, activating the force controller and leading to oscillations. The peaks caused by the acceleration regularly reached values between 20 and 40 N. The final obtained settings reach accidental peaks of only 15 N in comparison; and provide a stable disturbance response most of the time with a wave decay of around 50%.

The final step of the controller implementation was to find appropriate controller parameters for the PID terms for both the speed and force controllers with tests during an actual massage path with a load. This was done by manual tuning, as no mathematical model of the system was previously calculated by the designer of the electrical system. The tuning of the parameters is done on an abstract level. Optimally, the controller should have a step response without much overshoot and with strong dampening for disturbances as accidental movement by the patient can cause strong disturbances. The rising time is not a priority for optimization in this case. For a case where minimum over and undershoot is desired, the common method of Ziegler-Nichols is not recommended as it obtains only a wave decay of a quarter that wouldn't be sufficient. The conflict of the application is that the force feedback loop in this machine isn't adequate for a differential term in the PID force controller as the process variable contains too much noise. Sudden changes in the value of the process variable make the differential term fluctuate strongly in its output,

resulting in it actually being counter-productive. This was confirmed with tests showing that the introduction of a differential term to the force controller leads to instability. Due to this fact, the controller is limited in its dampening value. Regrettably, the manual tuning of the controller values wasn't conclusive as the test were unexpectedly interrupted by hardware complications.

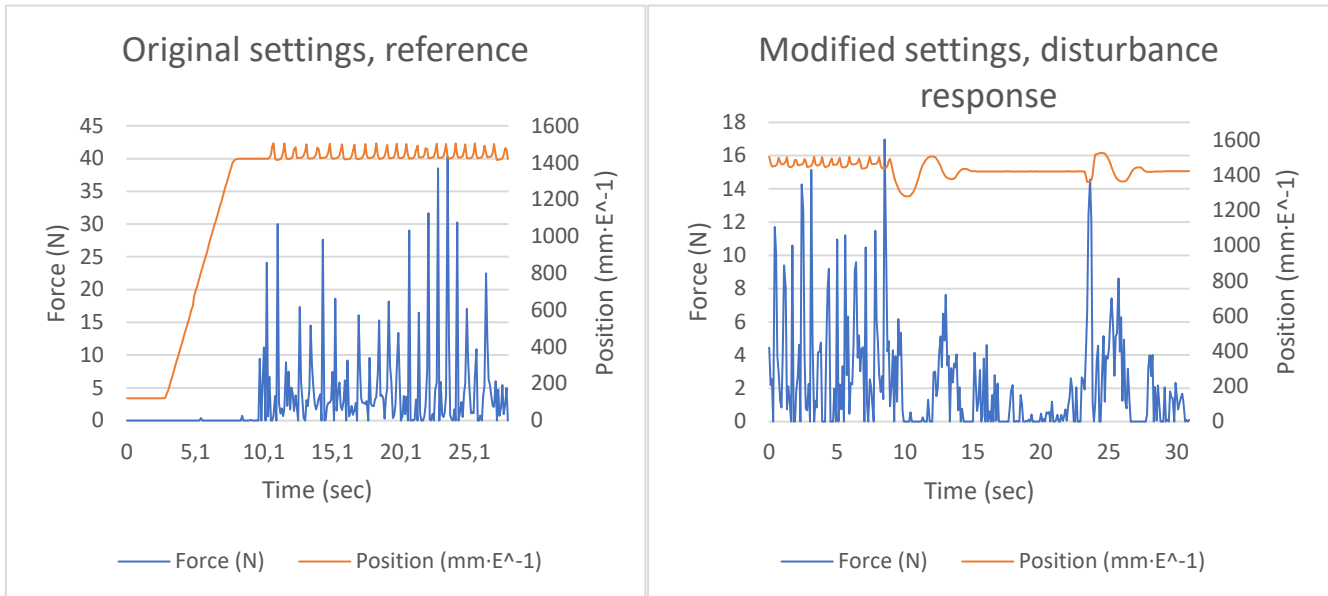


Figure 16: Test for dynamics without load (position = target + 12mm)

Alternative approaches to force control

In the extensive software package of this machine there many interlaced parts of its code that depend on each other to work as expected. Due to this reason, it proved very difficult to make considerable changes to the structure of the controller that would allow for a completely different control model. The main disadvantage of the current switching controller is that the response of the controller when switching between the two control laws cannot be adjusted properly and the resulting behavior can be quite difficult to analyze. Additionally, the switching logic can conflict with other operating modes of the machine giving as a consequence that getting it to work as planned, required much more testing and refining than planned. These were two aspects that resulted in substantial development effort during this thesis.

But switching controllers aren't the only possible way to program systems where process variables for both force and position must be controlled. The controllers in this general category can be called simultaneous controllers. The literature shows alternative methods that provide different benefits for different situations.

Simultaneous controllers are first categorized as either passive or active force controllers. Passive force controllers limit or restrict the force through mechanical components like springs or dampeners that have been calculated to provide the desired limiting effect. This method, naturally, is very dependent on the application and the form and function of the machine and doesn't foresee any kind of feedback loop for the force value. Active force controllers measure the force being applied and integrate it in some way into the controller. Active force controllers are further divided into direct and indirect controllers. Indirect controllers use the force measurements as an auxiliary value

and do not have an actual force feedback loop. Direct simultaneous controllers have a setpoint and loop for both the position and the force.

1. Passive force control
No force measurement. Force restricted mechanically with springs or dampeners.
2. Active force control
Force measurement integrated into the control law.
 - 2.1. Indirect force control
No force feedback loop. Force measurement is auxiliary.
 - 2.1.1. Impedance control (a.k.a. virtual compliance)
 - 2.1.2. Admittance control
 - 2.2. Direct force control
Uses setpoints for both force and position.
 - 2.2.1. Hybrid control (a.k.a. switching control)
 - 2.2.2. Inner/outer control
 - 2.2.3. Parallel control

Table 11: Classification of simultaneous force/position controllers [15] [16]

Indirect force control requires modelling the system mathematically. In virtual compliance a virtual (non-existing) spring and dampener are included in the model between the target position and the actual position of the effector. The parameters of these virtual components can be tweaked to obtain a desired behavior as the controller will compensate for the non-existing forces of these components. This results on an increasing force as the effector approaches its target position. Eventually the virtual resistance prevents the effector from moving further. This method can effectively be used to limit the force that a robot can exert on a surface while controlling its position. This limitation, though, will be a constant value. Opposite to impedance control is admittance control, where the effector is heavily dampened until a force is applied on it, allowing its movement. This method is relevant for other applications, like collaborative robotics.

In inner/outer control the force controller is located at the outermost layer of the cascade controller and its output acts as the setpoint for the position controller. Both controllers are provided an actual measured process variable to calculate the error. Finally, there is parallel control where both controllers act simultaneously and feed into the output of the system. In this case the force controller is tuned in a way that it dominates over the position when necessary.

An application like this massage bed could be best suited for inner/outer control, where a force controller with slower dynamics can output directly into the already accurate position controller. In cascade control each subsequent layer becomes more dynamic, with shorter rising times and less dampening. As currently programmed, the force controller outputs to the current controller, the innermost layer of the cascade controller. This means that the slowest controller of the system is interfacing with the fastest one. The position controller could also benefit from a feed-forward term setting a standard height target. Figure 17 shows the block diagram of the proposed controller.

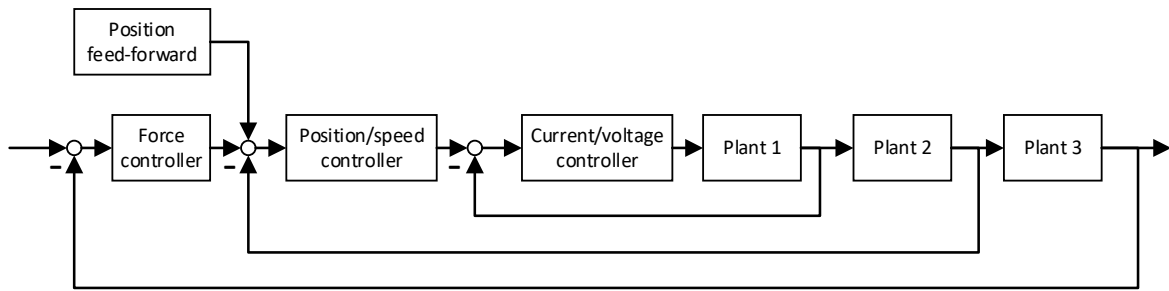


Figure 17: Proposed inner/outer controller

4.1.4 Microprocessor limitations and solutions

When introducing the new functionality to the processor it became immediately clear that its bottleneck and main limiting factor isn't its processing speed but the amount of available RAM memory. Most of the RAM has already been allocated to either storing the code of the program or as buffer for the telegrams. If the program is planning to use too much memory or its functions are too big to fit in the RAM, the compiler will fail to compile.

Luckily this type of processors offers the programmer complete freedom in the allocation of the RAM memory with the so-called linker command file. This file allows for manually specifying the size and address location for each and all sections for data storage. Several considerations must be taken when programming code for a processor with very limited RAM. First, is that some functions that are normally stored in the non-volatile FLASH memory of the chip must be loaded onto the RAM to increase performance. The difference in performance from this is very noticeable. Code running from it will run remarkably faster. Second, is that not all variables are treated equally by the compiler. Local undeclared variables used only inside a function or the `main()` are assigned to the much smaller ".stack" memory space while global variables declared outside of `main()` are assigned to the much bigger variable RAM section. The RAM is divided into two main sections. Starting at address 0x0 there is a small block of length 0x800,²⁷ where RAMM0 and RAMM1 are assigned for a total size of 2 Kw [17].²⁸ Further on, at address 0x8000 starts the main RAM comprising sections RAML0 to RAML7 with a total length of 0x8000 and a total size of 32 Kw.

The main use of the different sections is as follows:

- RAMM1: local variables ".stack"
- RAML0: functions that must be loaded on to ram "ramfuncs" and buffer VARRAML0
- RAML4: global variables ".ebss" & special section DMARAML4 for DMA
- RAML7: special section DMARAML7 for DMA

The existing code has been quite optimized in this aspect using the smallest possible variable size necessary for each variable, but some further improvements can be made. The code original code

²⁷ A number starting with 0x represents a number in hexadecimal base. In this case the addresses and section lengths are expressed in hex.

²⁸ 1 Kw (Kiloword) = 2 KB (Kilobyte)

was compiled years ago and in the meantime the compiler provided by Texas Instruments has received several updates. These offer better performance and/or better code size depending on the chosen compilation settings. The original compiler version of the PWM controller was 6.1.2 and the current version at time of writing is 18.1.5.

The final memory mapping after compilation can be found in “PWM_Regler.map” inside each build folder. Using this information, the different compiler versions and settings were compared. Just by updating the compiler, the size of the “ramfuncs” was reduced by 11%. Additionally, the new compiler version offers the option to further optimize the code for performance. This option was tested and rejected as it increased back the size of the “ramfuncs” by 54% while having no measurable change in the performance of the “Regler()” controller routine (compared to the benchmarks of chapter 3.2.1). Final adjustments were made by manually shifting the memory sections and allocating unused space to more important sections. These changes as a whole made it possible to continue expanding the program with the current hardware.

4.2 Main program

The main program takes care of coordinating all the actions to be performed by the components of the system as well as performing the necessary pre-processing and calculations that offload processing time and memory usage from the axes’ controllers. To the interest of this project were the routines used to load, process and transmit message paths as well as the receival, analysis and storage of scans’ information.

The main principle that was followed during the modifications of the program was to keep the implemented solutions as simple as possible and have them only modify the minimum amount of functions and data structures. This reduced the chances of involuntarily and unexpectedly affecting the functionality of other routines of the program as many parts of the program rely on common code to perform their actions. With this principle in mind, the best approach to deliver the desired force values to the axes’ controller was determined to be one that used as much as possible of the already existing communication infrastructure between the controllers and the main program. The chosen solution uses the telegrams that originally sent the values for the vertical position and replaces those values with the values for the desired force at each coordinate point that can be loaded from the message path file.

The message bed now has three different operation modes and can either utilize custom or standard (programmatically generated) message paths with or without custom force profiles. By changing the pre-compiler settings in the main program’s header file: “main_program.h” and recompiled as mentioned previously, in chapter 2.2.1: Loading message paths. How the combination of settings leads to the different modes and which combination of setting are not possible is explained in Flowchart 2 in Appendix E: Program flowcharts. When transmitting force values the force takes the place of the previously transmitted d3 vertical position as is isn’t utilized in force control mode. The communication between the main program and the A3 vertical controller occurs over telegrams where the main program signalizes its intent for an operation mode and the controller switches

accordingly. The error handling was updated on both sides to handle the different failure modes that may occur and reset the operation mode to the traditional control if the message was interrupted by any means. Otherwise the controller is completely agnostic to the type of values being received and may wrongly interpret force values as position coordinates leading to abrupt and dangerous movements.

4.2.1 Message path file format

The message paths are stored in a simple tab-delimited text file. It contains a header with information about the used body markers, then information for message intensity in the case of a kneading message and finally two or three columns of values depending on the path containing force values or not (Figure 18, left).

Rows one to three contain the coordinates of the markers that serve as fixed points of reference for the automatic path to body adjustment. The order of the coordinates is x, y, z and they are given in mm. Row four specifies which body markers are meant by the previously listed coordinates with 1 meaning used and 0 meaning not used. The order of the markers is from left to right: KS, SS, HS, SL, SR, HL, HR. These abbreviations respectively mean: head-sagittal, shoulders-sagittal, hips-sagittal, shoulders-left, shoulders-right, hips-left and hips-right. Rows five and six are only present for kneading massages and specify the orientation and intensity of the swashplate of massage tool MK1 [1, p. 104]. The following rows and until the end are the message coordinates (mm) and the force (N). These are from left to right: coordinates for the y and z position and finally, the force. The time interval between each point in the message path is set at 100 ms as shown in Table 12.

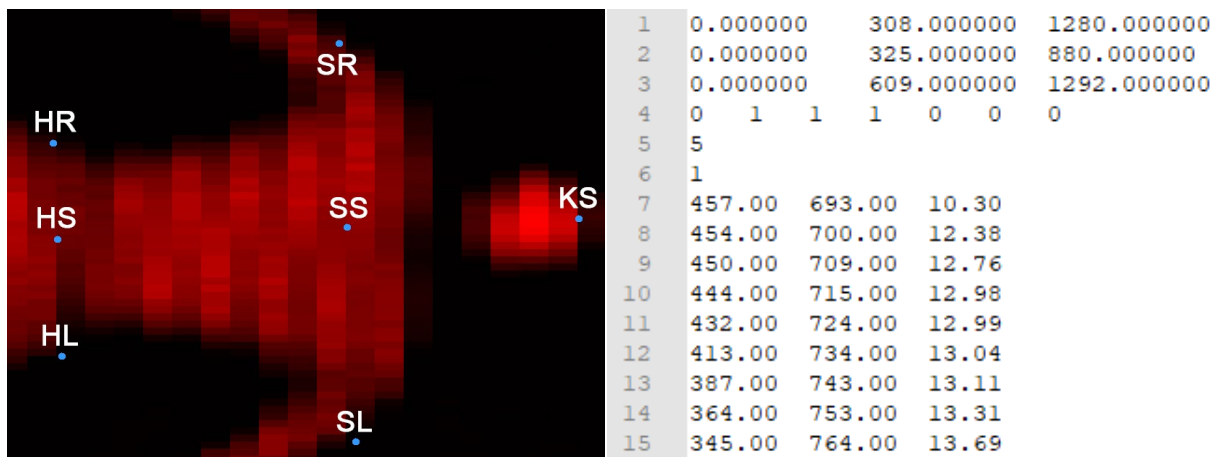


Figure 18: Body markers shown over scan of patient (left), exemplary message file for a kneading message path (right)

4.3 Tablet interface

To facilitate the creation of custom message paths that can be adapted to the needs of each patient and the desired techniques of each physician it was necessary to create a new interface that made it easy for non-technical personnel to create message paths. This interface should be easy to understand at first glance, easy to use and at the same time provide a good functionality to obtain the desired behavior of the massage tools.

To make this interface intuitive to the action of performing a massage it was decided to use a touch tablet with a pressure-sensitive pen. This would speed up and simplify the process of creating the trajectory for the massage tool while at the same time having the added value of an additional input for the physician, the force.

The most notable company that produces this kind of digital pressure sensitive tablets is Wacom. Their models vary strongly in capabilities including ones with or without screen; with or without integrated PC and with more, or less, sensing capabilities that allow to measure additional values such as tilt of the pen in two directions, distance to surface and rotation. The model that was available was the Intuos Pro. It is a model without screen and with full pen measuring capabilities (besides rotation), featuring 2048 levels of pressure sensitivity and 60 degrees of tilt. It connects to the PC as an input device and takes over the movement of the mouse when the pen approaches the tablet (direct contact isn't necessary for mouse movement). It is also otherwise impervious to hand touches, therefore making it easier to draw on it more naturally as if it were on a sheet of paper. The keyboard and mouse can still be used in conjunction for additional input inside the program.

In our use case, only the position of the pen on the tablet and the pressure value were relevant as the other values such as tilt and rotation of the massage tool are calculated programmatically and automatically. To obtain these values, Microsoft Windows has a library available in the form of a DLL that works with most kinds of touch tablets and can be called from any C/C++ program, `wintab32`. DLL stands for dynamic-link library and is a distributable piece of precompiled code that gathers many functions that may be used by other programs. This allows for distribution of code for collaboration without making the original source code visible to all users. A DLL usually also includes an `.h` or `.hpp` header file that describes the functions and structures used in the library and what input they need.

As the information from the tablet is needed inside LabVIEW, the first attempted solution was to use the included VIs in LabVIEW that allow using DLL functions directly. This turned out not to be a practical solution because the Wintab API uses a lot of custom data structures that aren't easily transferred from and back to LabVIEW. The alternative solution in a case like this is to create a so-called wrapper-DLL. This new library calls the DLL, that was originally intended to be used in its C++ code, natively and therefore can make full use of the structures of the API without any complications. This new library has some functions that encompass all the needed functionality in the minimum amount of necessary functions for the use of the tablet and with simplified inputs. This new library is then called from LabVIEW and returns the pressure values of the pen.

The interface was built around the use of an image frame element. Inside the frame any kind of image can be displayed, and its contents can be drawn on with additional information. The main image displayed on this frame on program start is the last capacitance scan loaded by the main program. After executing a scan from the LabVIEW interface the massage path creation interface can be accessed from it (see Figure 6 in chapter 2.2.1) and it will display the scan that was just loaded or performed. This scan comes from the temporary scan file mentioned in chapter 2.2.1:

Scans and analysis. Additionally, the body markers calculated by the body analysis algorithm will be displayed over the scan, for orientation when drawing the massage path (as shown in Figure 19).

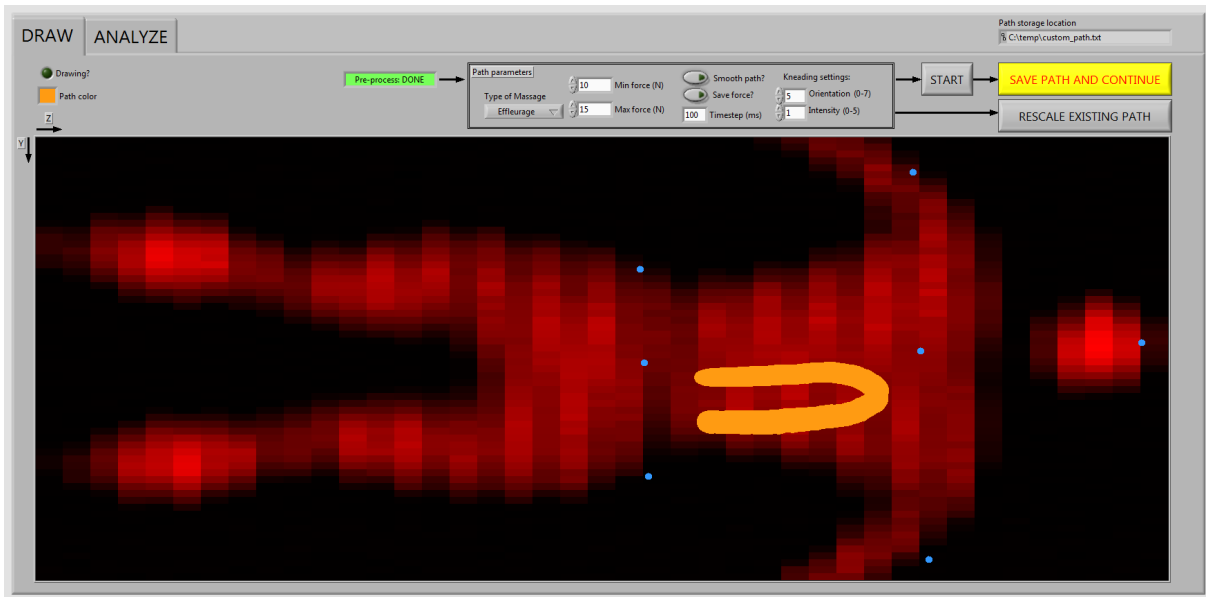


Figure 19: Massage path creation interface (draw tab), path in orange and body markers as blue dots

After the pre-processing is completed, the settings for the desired massage must be chosen. First of all, the type, either effleurage or kneading. If the latter was chosen then additionally the intensity and orientation values (as explained in the previous chapter) must be entered. The minimum and maximum forces correspond to the minimum and maximum detectable pressure of the pen. These can be set to any value between 10 and 200 N and the program will scale the pressure accordingly to generate the force value. The timestep specifies the fixed time interval that separates one coordinate from the next one. This time interval varies depending on the type of movement being executed. The standard time interval for massages is 100 ms. The time intervals for the different routines are shown in Table 12. If “Save force” was activated, then the massage file will include a third column with force values; and if “Smooth path” was chosen, the y-z coordinates will be smoothed by a moving average with a 3-samples window to even out its movements. After finalizing all settings, the start button must be pressed to enable the drawing surface.

Type of routine	Time interval (ms)
General	100
MK1 (kneading)	100
MK2 massage (effleurage)	100
MK2 scan	50

Table 12: Time intervals for the massage paths



Figure 20: Intuos Pro touch tablet

The speed at which the massage is drawn is directly related to the speed at which the massage will be performed, to make it easy to understand. The program records the position and pressure of the

pen at a high poll rate. These values are then filtered out at multiples of the chosen time interval to create the final path file. After drawing the path, it must be saved by pressing the “save path” flashing button. The results can be then observed in the “analyze” tab as shown in Figure 21. As an alternative to creating a new message, the min/max force setting can be used to rescale the force of a previously drawn message path to new values. The message path file to be modified must be stored as “C:\temp\custom_path.txt”.

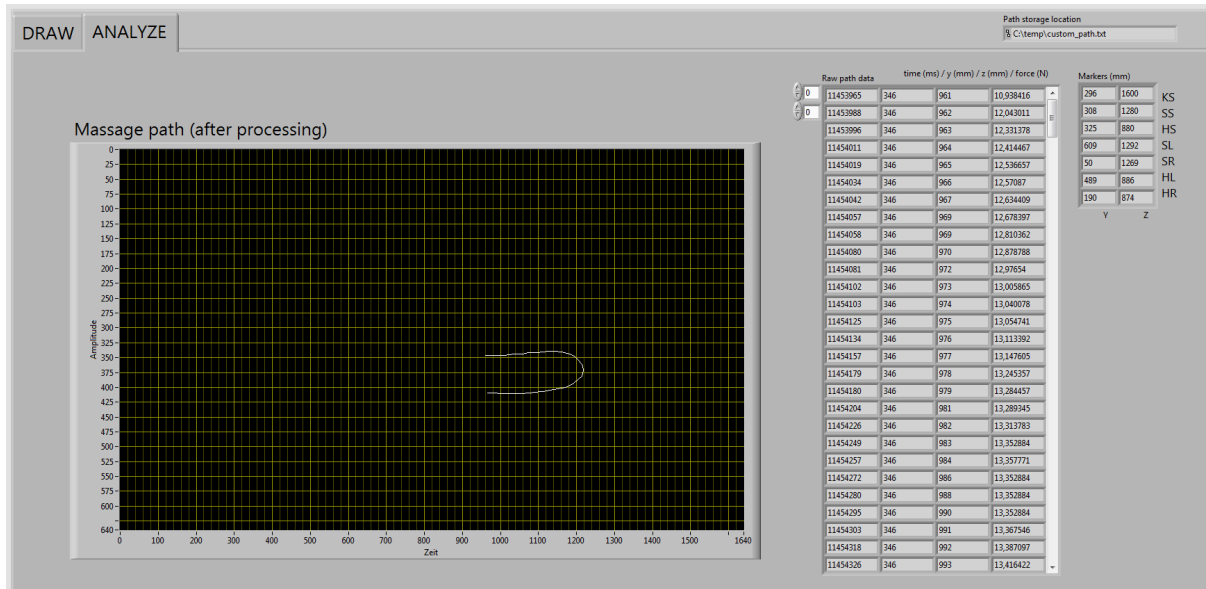


Figure 21: Message path creation interface (analyze tab)

Annex: dependencies and requirements of the program

For future reference, these are the required files to compile and/or run the message path creation interface:

- Name of Wrapper DLL:
“Wintab32 DLL Wrapper.dll”
- Dependencies for re-compiling:
“WINTAB.H” & “PKTDEF.H” (provided by Wacom)
“UtilsExpanded.cpp” & “UtilsExpanded.h”
(originally provided by Wacom and then customized)
- Dependencies for running:
Installed with Wacom drivers: “wintab32.dll”

5 Conclusions

The goal of this thesis can be divided into two main objectives. The first part was the development of an improved user interface for drawing massage paths with a touch tablet and the second was the expansion of the communication and controlling capabilities to achieve force control. These two developments can be utilized independently for different purposes.

The first objective was very satisfactorily achieved, fulfilling all the initial specifications. The obtained program can, with ease of use, draw any desired massage path, and makes use of all the relevant information (scans and body markers) provided by the main program to facilitate drawing the path. The obtained path can afterwards undergo post-processing to improve its smoothness or scale the specified force. The program was programmed independently of the main HMI and therefore retains its freedom of customizability. The interfacing with the other software elements was designed in a straightforward way to simplify eventual future development. This new interface fills a very crucial gap in the concept of the massage bed, connecting the therapists to the program.

The second objective encountered some difficulties in its development. The force controller was in some respects limited by the existing design. As the key components of the machine have been already specified by previous work on the project, these could not be changed and had to be worked around. In contrary to the initial concern, the Delfino microprocessor proved to be fast enough to execute the desired algorithms but the existing self-implemented sensorics turned out not to be totally adequate for the implementation of force control. The considerable noise observed in the force measurement limits the possibilities in controller design by making differential terms in the PID not applicable, therefore not allowing for good damping. Additionally, the sensor bus had very frequent unreliability issues that interrupted all movements of the machine for safety reasons. As substantial development time was being used to troubleshoot sensor issues, these problems led to halting the search for new parameters for the force controller.

The development of the new force control mode was brought to completion despite these issues. It includes the transmission of the force values from the drawn path all the way to the vertical axis controller through telegrams and a reworked controller logic that allows for two different modes of force control: fixed force or custom force profile. The only aspect that would still require future work would be the adjusting of the controller parameters.

For an eventual continuation of this project in the future, some recommendations can be suggested based on observations made during this thesis. When tackling a completely new development, a balance should be sought between retaining the customizability of every component by designing in-house and reliability by buying off-the-shelf products. It can be recommended that, if the development of a specific component is not the focus of the research itself, that it be acquired as an off-the-shelf product. This improves the reliability of the implementation, reducing errors, development time, and practically extending the service life of the prototype. Some component whose market alternatives could improve the reliability of the machine include the force sensor that

could be replaced by a regular load cell with integrated amplifier to transmit a more reliable signal and regular rotary encoders to tackle the sensorics problems.

The development of this project teaches many lessons about working on complex mechatronic projects where contributions by many researchers are involved. The documentation of both the mechanical design as well as the utilized code and algorithms is of the utmost importance. In this regard, an effort was made to improve the knowledge management of the whole process. The written code was thoroughly commentated, previously unavailable wiring diagrams of the complete massage bed were drawn, and all the software elements used by the many different processors were collected and organized. These contributions, in addition to the integral description of the system written for this thesis, will facilitate future contributions by any new researchers.

6 Appendices

Appendix A: How to execute a message

To execute a message from zero, follow these steps:

1. **Choose the control method.** Decide if position, fixed force or force profile control will be used by changing the pre-compiler settings in “main_program.h”.
 - USE_FORCE_CONTROL
 - USE_PATH_WITH_FORCE
 - MAX_FORCE (is the setpoint in case of fixed force control)
(see Message operating modes in Appendix E: Program flowcharts)
2. **Decide if new a scan will be performed or an existing one will be sideloaded.** This can also be changed in the pre-compiler settings in “main_program.h”.
 - LOAD_C_SCAN_TESTFILE
 - LOAD_X_SCAN_TESTFILE
 - PERFORM_C_SCAN
(see Table 2 and its corresponding explanation)
3. **Decide if a custom message path will be used,** by changing the following pre-compiler setting.
 - USE_CUSTOM_PATH
4. **Mount the message tool:**
 - MK1 fore kneading
 - MK2 for effleurage and/or scanning
5. **Turn on the power supply.** Turn on and verify the settings (42 V with current limits of ~10.25 A and ~0.75 A for PW1 and PW2 respectively).
6. **Start the LabVIEW HMI.**
7. **Press “Alle referenzieren” to zero all axes.**
8. **Execute or sideload the scans:** “Dist.Scan” first and then “Cap-Scan”. Actually performing the scan will take several minutes. Wait for the message carriage to stop completely at the origin before proceeding.
9. **(optional) Open the message creation interface and draw the desired path.** This step can be skipped and still load a custom path if the path is still present in storage.
10. **Load the message path** by choosing the message type and pressing “LADEN”. To load a custom path from the previous step, choose effleurage.
11. **Start the message** by pressing “Starte Massage”.

Appendix B: Location of project files and backups

Inside the main project folder:

- Software_package
 - Backups_Pfinsterwalder-Bergkichner_finalDelivery
 - HMI
 - HMI.vi
 - Main_Program
 - Microprocessors_sourcecode
 - BUS Schnittstelle Delfino (Main BUS)
 - InterBusTransceiver (Interbus transceivers, both)
 - PWM_Regler_Rom_isr V2.12 (Motor controllers, all equal)
 - MSP430F5510MK_Sensorik (MK2)
 - MSP430F5510MK_Steuerung V1.3 (MK1)
 - Microprocessors_flashdumps
 - Main BUS.bin
 - Interbus MB1.bin
 - Interbus MB2.bin
 - PWM controller.bin (all equal)
 - Actual_Abbate_finalDelivery
 - HMI
 - HMI.vi ←UPDATED
 - Message path creation interface.vi ←UPDATED
 - Main_Program ←UPDATED
 - Microprocessors_sourcecode
 - PWM_Regler_Rom_isr V2.13 (Motor controller A3) ←UPDATED
 - MSP430F5510MK_Sensorik (MK2)
 - MSP430F5510MK_Steuerung V1.3 (MK1)
 - BUS Schnittstelle Delfino (Main BUS)
 - InterBusTransceiver (Interbus transceivers, both)
 - PWM_Regler_Rom_isr V2.12 (Motor controllers, A1,A2,A4-A6)

Appendix C: How to directly flash a Delfino without CCS

The flash memory of the microprocessors can be accessed directly to execute read or write operations. This way code can be backed up without access to its source code and later restored one to one to its original state.

The general flash memory is divided into eight sections named FLASHA to FLASHH. Each section has a length of 0x8000 and every address stores 2 bytes.²⁹ This results in a size of 64 KB per section and a total size for the whole Flash of 512 KB [17]. The Flash memory starts at address 0x300000 with section FLASHH and ends at 0x33FFFF with section FLASHA. This information about memory allocation to each of the sections can be found in the linker command file of each program, “F28335.cmd”, that specifies how all memory, RAM and Flash, is allocated.

The whole memory can be read and saved into a .bin-file of 512 KB that stores pure binary data. This file can then be used to write again to flash and overwrite any existing information. The program used to do so during this project was Uniflash (v4.6)³⁰. To connect to the microprocessor in question one must use a JTAG emulation probe. The probe used was the Blackhawk USB200 JTAG Emulator (BH-USB-200). The JTAG pins of the controller boards are accessible without opening the enclosure. For the main bus and inter-bus transceivers the enclosure must be opened to access the JTAG pins.

Once in Uniflash a target configuration file must be chosen to tell the program how we want to reach the processor (as an alternative, the type of processor and JTAG probe can be selected from the lists). Target configuration files have a .ccxml-file type and are located inside the folder of the source code for each microprocessor, by example: “PWM_Regler.ccxml”.

- To read:
In the Memory tab, press **Read Target Device** to show the memory of the microprocessor. To export the complete program stored in the flash memory of the microprocessor press **Export** and enter address 0x300000, number of bytes 0x40000 and export as a .bin-file.
- To write:
In the Program tab, select a flash image to write to the microprocessor. This file can either be a previously stored binary .bin-file or a compiled .out-file (stored inside the build folder of a CCS project); by example:
“PWM_Regler_Rom_isr V2.12\Debug Modern\PWM_Invertercontroller.out”.

²⁹ A number starting with 0x represents a number in hexadecimal base. In this case the addresses and section lengths are expressed in hex.

³⁰ Note: Uniflash v4.0 or higher doesn't provide native support for Blackhawk probes. See [25] for details on compatibility.

When writing a binary file, an address must be additionally provided as this file contains no information about its original location. For a complete Flash dump this address would be 0x300000.

Press: **Load Image** to write to the microprocessor.

- To verify:

To verify the program present in the flash memory of the microprocessor without making any changes to it, a flash image can be chosen in the same way.

Press: **Verify Image**.

If the program is exactly the same as the one loaded, the verification will be successful.

Warning: The checksum cannot reliably verify a program and should not be used for that purpose!

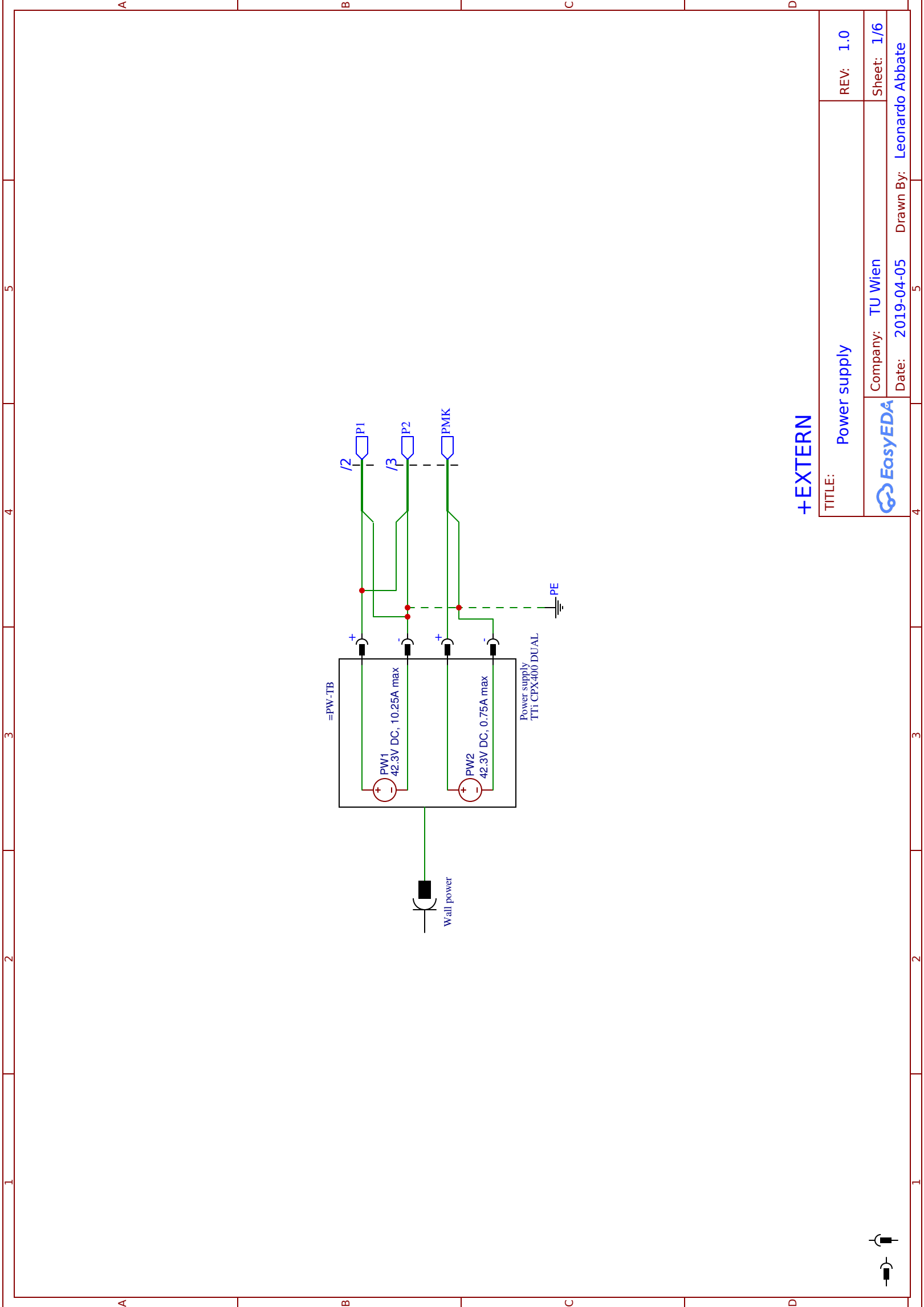
Appendix D: Wiring diagrams

Table 13: List of components


Component	Identification	Page
Power supply, TTi CPX400 DUAL	=PW-TB	1
PC	=PC-KE	2
Bus controller	=BUSCON-KF1	2
FSB LED	-PFA	2
FSB Jumper	-QB	2
Axis controller	-A1	3
Axis controller	-A2	3
Axis controller	-A3	3
Axis controller	-A4	3
Axis controller	-A5	3
Axis controller	-A6	3
Motor	-MA1	4
Motor	-MA2	4
Motor	-MA3	4
Motor	-MA4	4
Motor	-MA5	4
Motor	-MA6	4
Magnetic rotary encoder	-BGF1	4
Magnetic rotary encoder	-BGF2	4
Magnetic rotary encoder	-BGF3	4
Magnetic rotary encoder	-BGF4	4
Magnetic rotary encoder	-BGF5	4
Magnetic rotary encoder	-BGF6	4
Sensor bus controller	=SB-KE1	4
Sensor bus controller	=SB-KE2	4
Sensor bus controller	=SB-KE3	4
Sensor bus controller	=SB-KE4	4
Sensor bus controller	=SB-KE5	4
Sensor bus controller	=SB-KE6	4
Limit switch	-BGA1.1	4
Limit switch	-BGA1.2	4
Limit switch	-BGA2.1	4
Limit switch	-BGA2.2	4
Limit switch	-BGA3.1	4
Limit switch	-BGA3.2	4
Limit switch	-BGA5.1	4
Limit switch	-BGA5.2	4

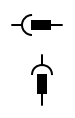
Vertical position switch	-BGA5.3	4
Strain gauge	-BW3	4
Processing unit, Texas Instruments MSP430F5510	=MK1-KE7	5
Processing unit, Texas Instruments MSP430F5510	=MK2-KE8	5
SIM sliding connector	-X1	5
Motor, Faulhaber 1717A024S R IE2-16 15A 5.3:1	-M7.2	5
Motor, Faulhaber 1717A024S R IE2-16 15A 5.3:1	-M7.2	5
Motor, Faulhaber 1717A024S R IE2-16 15A 5.3:1	-M7.3	5
Motor controller, Texas Instruments DRV8808	-KE7.1	5
Motor controller, Texas Instruments DRV8808	-KE7.2	5
Motor controller, Texas Instruments DRV8808	-KE7.3	5
Rotary encoder	-BGF7.1	5
Rotary encoder	-BGF7.2	5
Rotary encoder	-BGF7.3	5

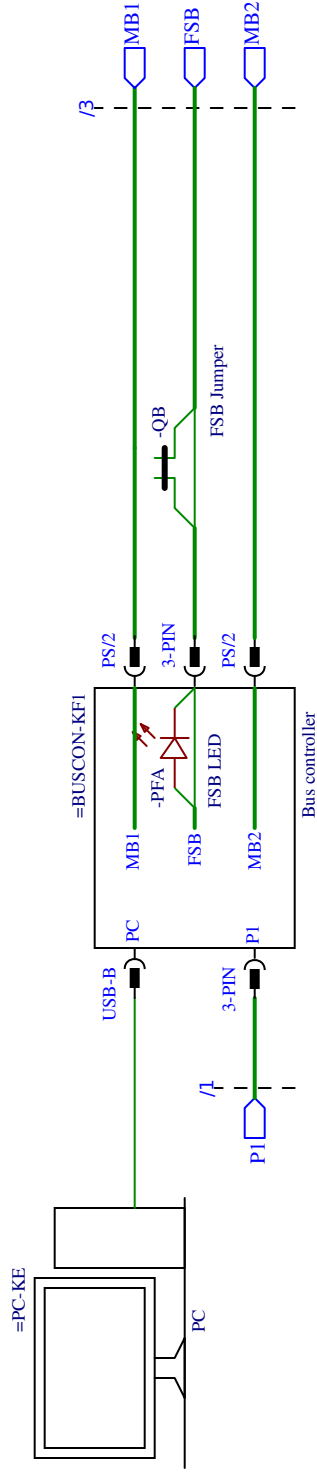
Identifications according to: DIN EN 81346 [18].



+EXTERN

TITLE:	Power supply	REV:	1.0
		Company:	TU Wien
		Date:	2019-04-05
		Drawn By:	Leonardo Abbate
		Sheet:	1/6





+EXTERN

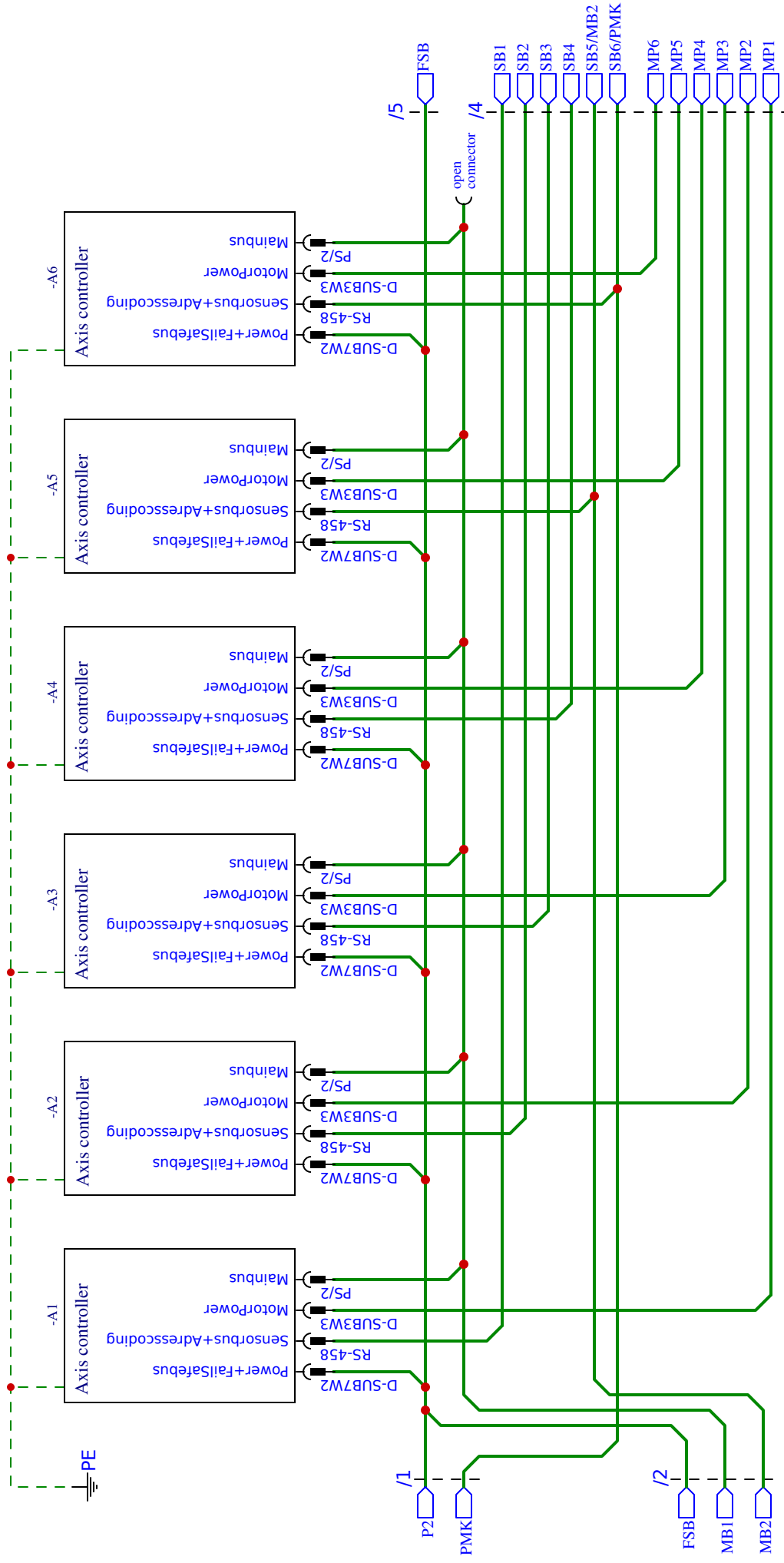
TITLE:	PC-connection	REV:	1.0
		Sheet:	2/6
		Date:	2019-04-05
		Drawn By:	Leonardo Abbate



Company: TU Wien

Date: 2019-04-05

Drawn By: Leonardo Abbate



+EXTERN

TITLE:

Axes' controllers

REV: 1.0

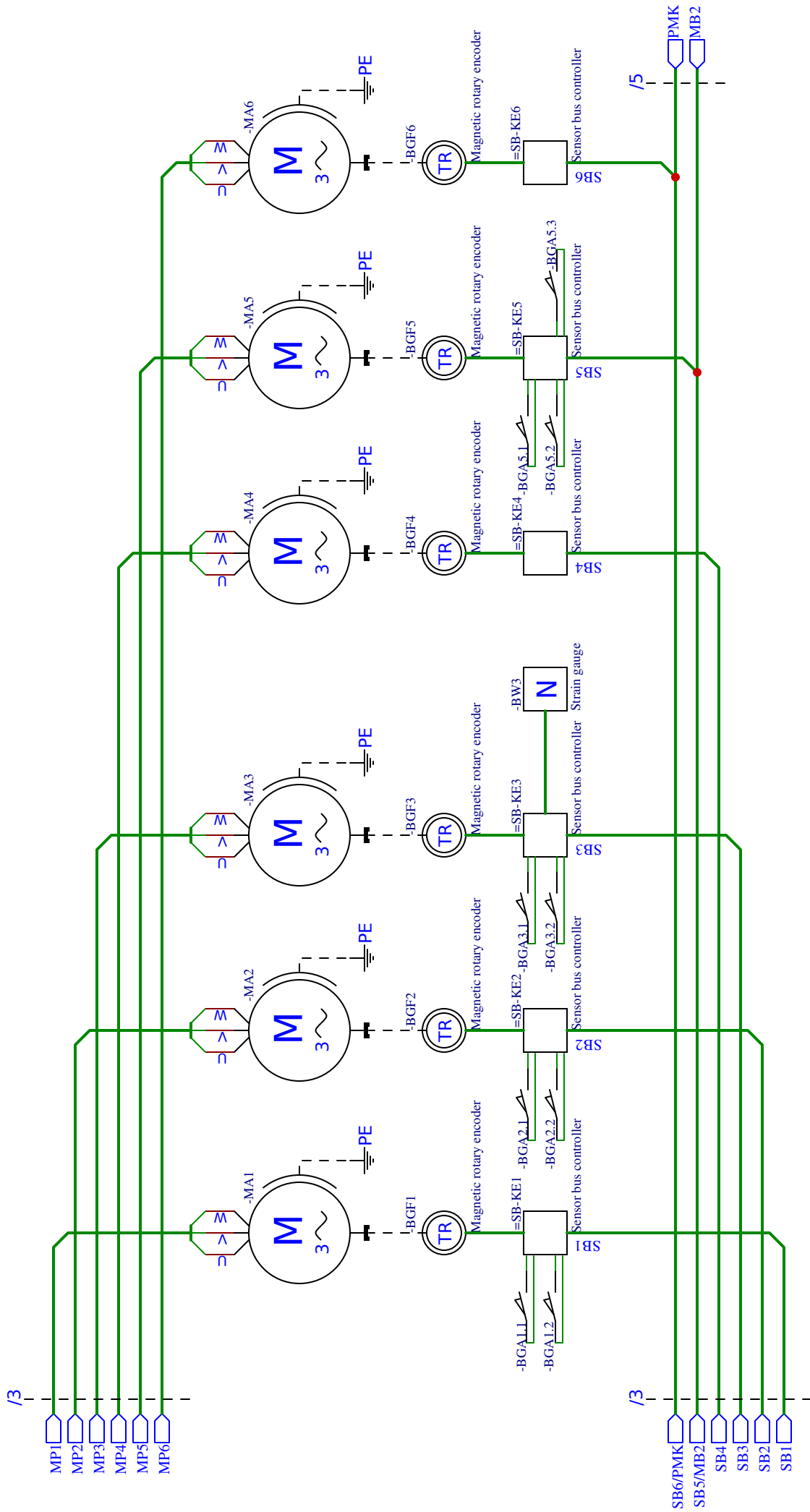


Company: TU Wien

Sheet: 3/6

Date: 2019-04-05

Drawn By: Leonardo Abbate

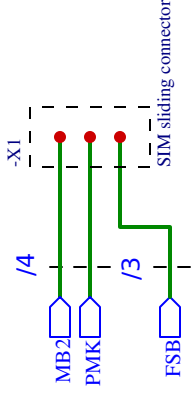
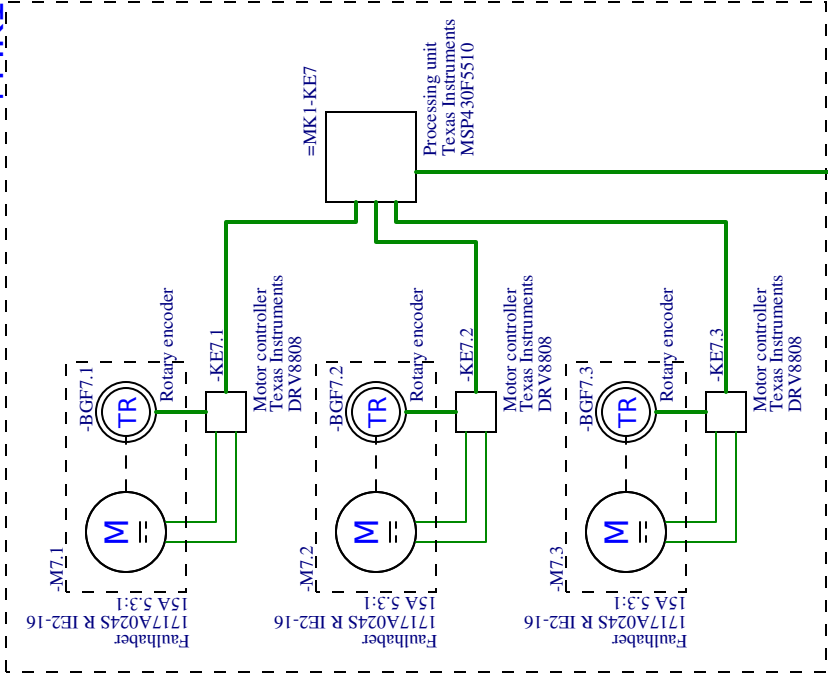


+BED

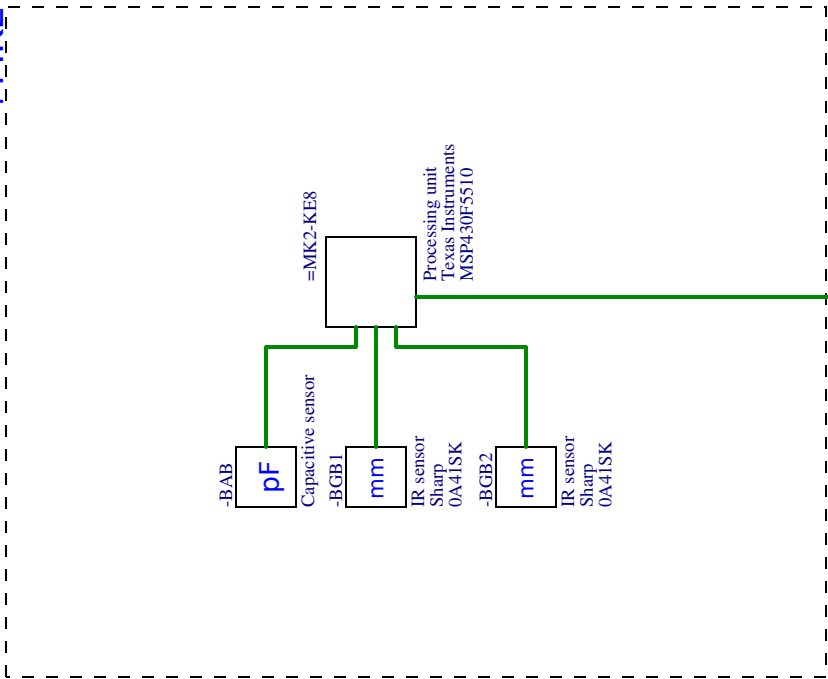
TITLE:	Motors and sensors	REV:	1.0
Company:	TU Wien	Sheet:	4/6
Date:	2019-04-05	Drawn By:	Leonardo Abbate



+MK1

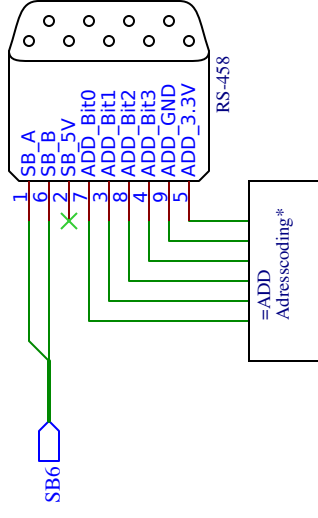
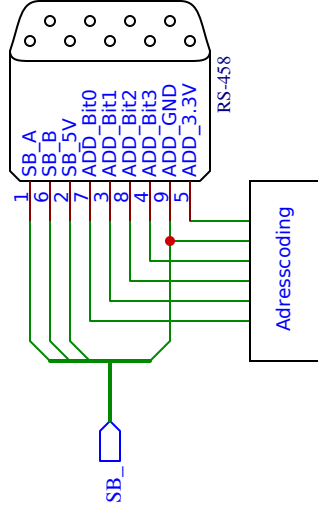


+MK2



+BED

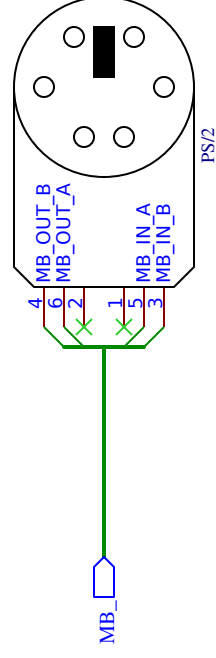
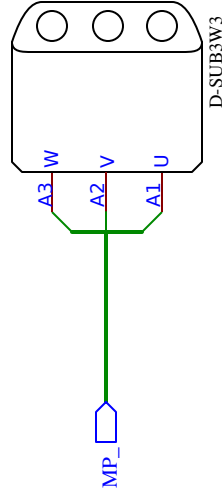
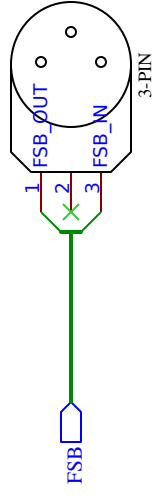
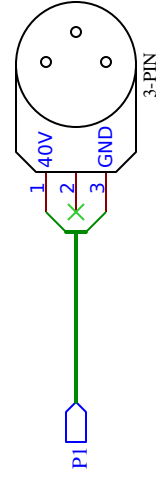
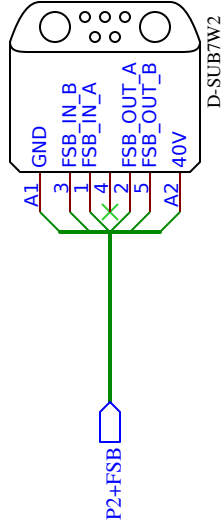
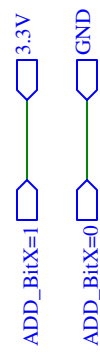
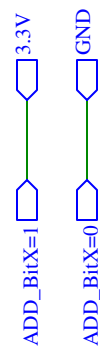
TITLE:	Message heads	REV:	1.0
		Company:	TU Wien
		Date:	2019-04-14
		Drawn By:	Leonardo Abbate
		Sheet:	5/6



* =ADD isn't a physical component.
It represents instructions to connect the ADD_BitX pins.

	A1	A2	A3	A4	A5	A6
Bit0	1	0	1	0	1	0
Bit1	0	1	1	0	0	1
Bit2	0	0	0	1	1	1
Bit3	0	0	0	0	0	0

Addresscoding



+EXTERN

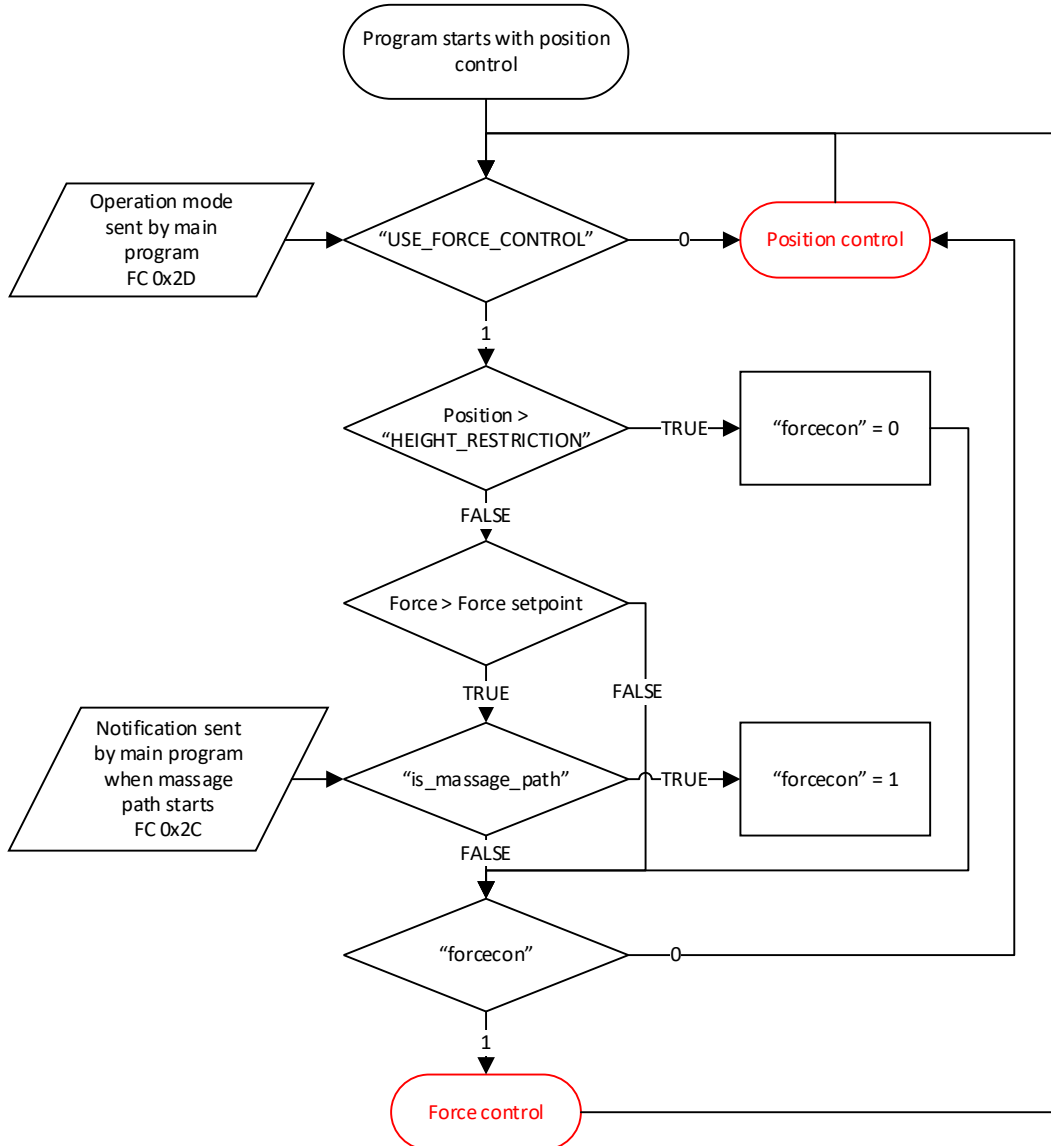
TITLE: Connectors	REV: 1.0
Company: TU Wien	Sheet: 6/6
Date: 2019-04-09	Drawn By: Leonardo Abbate



Appendix E: Program flowcharts

Controller switching logic

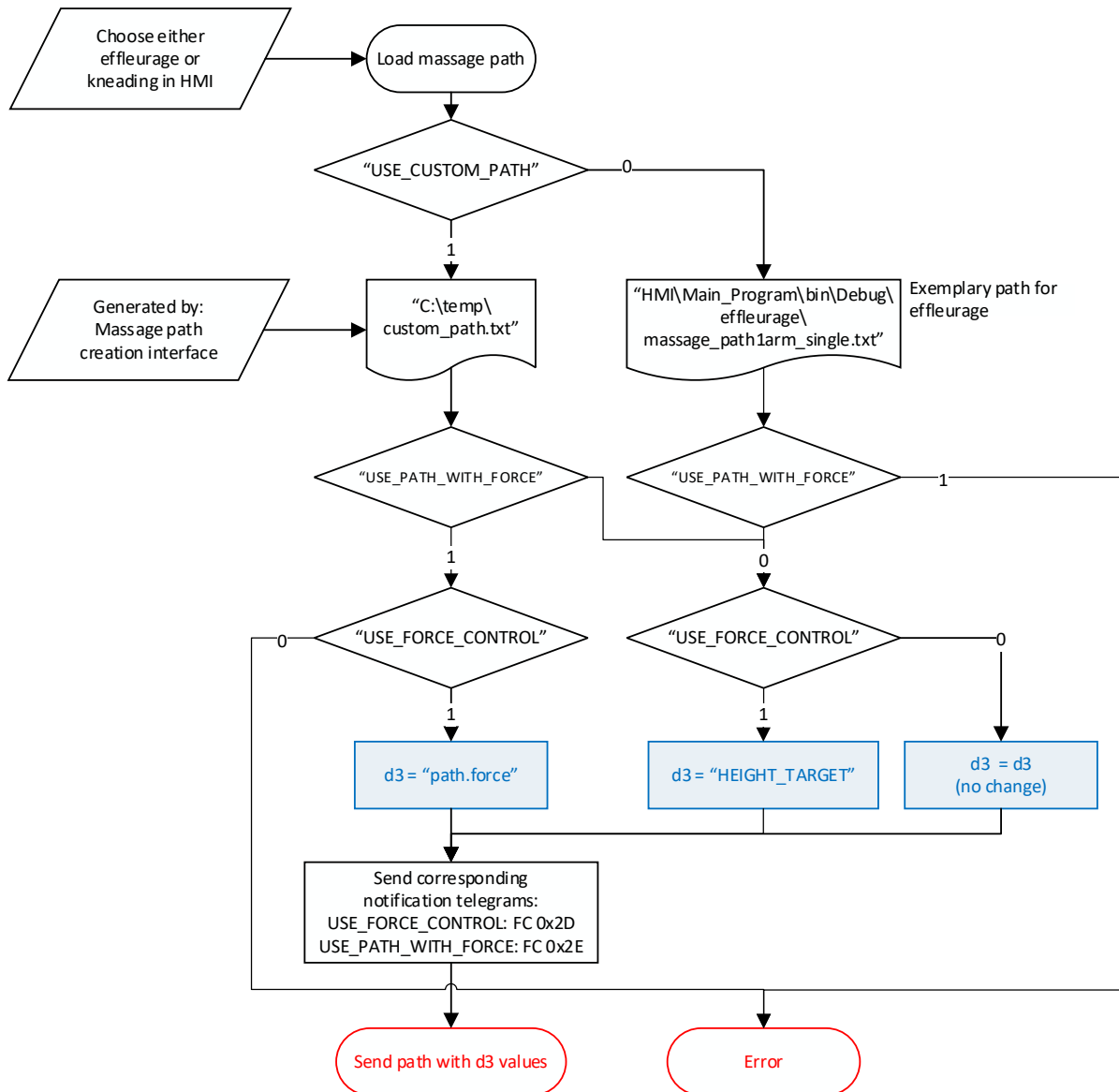
“PWM_Regler_Rom_isr\Regelung.c\Drehzahl_f_dq()”



Flowchart 1: controller switching logic (FC stands for function code)

Massage operating modes

“HMI\Main_Program\bed_main.cbp”



Flowchart 2: massage operating modes (FC stands for function code)

References

- [1] P. Finsterwalder, „Analyse und mechanisches Konzept therapeutisch wirksamer Massagebewegungen,“ TU Wien, 2015.
- [2] B. C. Kolster, *Massage*, Springer, 2006.
- [3] M. Clobert, T. L. Sims, Y. Miyamoto, H. R. Markus, M. Karawasa and C. S. Levine, *Feeling Excited or Taking a Bath: Do Distinct Pathways Underlie the Positive AffectxHealth Link in the U.S. and Japan?*, American Psychological Association, 2019.
- [4] T. Teramae, D. Kushida, F. Takemori and A. Kitamura, "Construction of an Intelligent Massage System Based on Human Skin–Muscle Elasticity," *Electronics and Communications in Japan, Vol. 94, No. 10; Wiley Periodicals, Inc.*, 2011.
- [5] C.-g. Kang, B.-j. Lee, I.-x. Son and H.-y. Kim, "Design of a Percussive Massage Robot Tapping Human Backs," *16th IEEE International Conference on Robot & Human Interactive Communication*, 2007.
- [6] P. Minyong, T. Miyoshi, K. Terashima and H. Kitagawa, "Exper Massage Motion Control by Multi-fingered Robot Hand," *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systemts*, 2003.
- [7] H. Ishii, H. Koga, Y. Obokawa, J. Solis, A. Takanishi and A. Katsumata, "Path generator control system and virtual compliance calculator for maxillofacial massage robots," *Int J CARS (2010) 5:77x84; Springer*, 2009.
- [8] P. Archer, *Therapeutic Massage in Athletics*, Lippincott Williams and Wilkins, 2006.
- [9] Texas Instruments, "TMS320C28x CPU and Instruction Set Reference Guide (SPRU430F)," 2015. [Online]. Available: <http://www.ti.com/lit/ug/spru430f/spru430f.pdf>. [Accessed 2018].
- [10] K. Bergkirchner, „Entwicklung eines mechatronischen therapeutischen Massagesystems,“ TU Wien, (not yet published).
- [11] F. Lei, F. Min and G. Min-min, "Study on Force Mechanism for Therapeutic Effect of Pushing Manipulation with One-Finger Meditation Base on Similarity Analysis of Force and Waveform," *Chin J Integr Med 2018 Jul;24(7):531-536*, 2018.
- [12] M. Gössl, „Entwicklung eine Motorsteuerung für ein mehrachsiges Aktorsystem,“ TU Wien, 2015.
- [13] Texas Instruments, "C2000 Real-Time Control Peripherals Reference Guide (SPRU566M)," 2018. [Online]. Available: <http://www.ti.com/lit/ug/spru566m/spru566m.pdf>. [Accessed 2018].
- [14] Texas Instruments, "Code Composer Studio User’s Guide (SPRU328B)," 2000. [Online]. Available: <https://www.eit.lth.se/fileadmin/eit/courses/eti121/References/ccs.pdf>. [Accessed 2018].
- [15] A. Owen-Hill, "Force Sensors in Robotics Research," *Robotiq*, 2016.

- [16] S. Chiaverini, B. Siciliano and L. Villani, "Force and Position Tracking: Parallel Control With Stiffness Adaptation," *Proceedings of the 1997 IEEE international Conference on Robotics and Automation*, 1998.
- [17] Texas Instruments, "C2000 Delfino Multi-Day Workshop," 2010. [Online]. Available: http://processors.wiki.ti.com/index.php/C2000_Multi-Day_Workshop. [Accessed 2018].
- [18] Deutsches Institut für Normung, *DIN EN 81346: Industrielle Systeme, Anlagen und Ausrüstungen und Industrieprodukte x Strukturierungsprinzipien und Referenzkennzeichnung*, 2010.
- [19] Texas Instruments, "Running an Application from Internal Flash Memory on the TMS320F28xxx (SPRA958L)," 2013. [Online]. Available: <http://www.ti.com/lit/an/spra958l/spra958l.pdf>. [Accessed 2018].
- [20] Texas Instruments, "F2833x Firmware Development Package Users's Guide," 2018. [Online]. Available: http://dev.ti.com/tirex/content/tirex-product-tree/C2000Ware_1_00_06_00_Device/device_support/f2833x/docs/F2833x_DEV_USE_R_GUIDE.pdf. [Accessed 2018].
- [21] Texas Instruments, "TMS320x2833x, 2823x Serial Communications Interface (SCI) Reference Guide (SPRUFZ5A)," 2009. [Online]. Available: <http://www.ti.com/lit/ug/sprufz5a/sprufz5a.pdf>. [Accessed 2018].
- [22] B. Reichert, *Massage-Therapie*, Thieme, 2015.
- [23] W. Wang, P. Zhang, C. Liang and Y. Shi, "Design, path planning improvement and test of a portable massage robot on human back," *International Journal of Advanced Robotic Systems July-August 2018: 1x11; Sage*, 2018.
- [24] P. Dutkiewicz and M. Michalek, "Impedance control with virtual compliance," *DOI: 10.1109/ROMOCO.2002.1177085; Poznan University of Technology*, 2002.
- [25] M. Brandstötter, S. Mühlbacher-Karrer, D. Schett and H. Zangl, "Virtual Compliance Control of a Kinematically Redundant Serial Manipulator with 9 DoF," *International Conference on Robotics in Alpe-Adria Danube Region*, 2016.
- [26] Blackhawk, "CCS UniFlash v4 - Blackhawk Support," [Online]. Available: <https://www.blackhawk-dsp.com/support/uniflashv4>. [Accessed 2018].
- [27] Deutsches Institut für Normung, *DIN EN 60617: Graphische Symbole für Schaltpläne*, 1997.

List of figures

- Figure 1: Massage movements for kneading (left) and effleurage (right)8
- Figure 2: Movement axes depicted over massage carriage (left) and bed (right)..... 10
- Figure 3: Massage tool MK2 with capacitance and IR distance sensors.....11
- Figure 4: Massage tool MK1 with swashplate and motors.....11
- Figure 5: Diagram of software and hardware connections.....12
- Figure 6: Functions of the HMI test interface 13
- Figure 7: Functions of the amplifier interface.....14
- Figure 8: Capacitance scan (top) and distance scan (bottom) 15
- Figure 9: Varying cycle times when benchmarking routines of the microprocessor25
- Figure 10: Recorded force and strain gauge (DMS) values for a static load (left) and manually induced current safety limit (right)30
- Figure 11: Recorded force and strain gauge (DMS) values for force limiting at 10 N (left) and FFT of the force values during the massage (right)32
- Figure 12: Recorded force and strain gauge (DMS) values for force limiting at 60 N (left) and FFT of the force values during the massage (right)32
- Figure 13: Superposition of the unmodified DMS values and the averaged ones 34
- Figure 14: Block diagram for the position controller 34
- Figure 15: Block diagram for the force controller..... 35
- Figure 16: Test for dynamics without load (position = target + 12mm)37
- Figure 17: Proposed inner/outer controller39
- Figure 18: Body markers shown over scan of patient (left), exemplary massage file for a kneading massage path (right).....41
- Figure 19: Massage path creation interface (draw tab), path in orange and body markers as blue dots 43
- Figure 20: Intuos Pro touch tablet 43
- Figure 21: Massage path creation interface (analyze tab).....44

List of tables

Table 1: Axes of the massage bed.....9

Table 2: Scans pre-compiler settings.....16

Table 3: Telegrams' structure.....17

Table 4: Relation between body-characteristics and massage comfort [4].....22

Table 5: Considered processors' specifications.....24

Table 6: Summary of data for the benchmarking.....26

Table 7: Summary of data for the static load measurements.....29

Table 8: Dynamic measurements' test setup information for reproducibility.....30

Table 9: Summary of data for the dynamic measurements.....32

Table 10: Summary of measurements for the improvement of the stability of the reference value 33

Table 11: Classification of simultaneous force/position controllers [15] [16]38

Table 12: Time intervals for the massage paths43

Table 13: List of components.....51

List of abbreviations

a.k.a.	Also known as
ADC	Analog to digital converter
API	Application programming interface
CCS	Code composer studio
CNC	Computer numerical control
DAC	Digital to analog converter
DAM	Direct memory access
DLL	Dynamic-link library
DMS	Dehnmessstreifen (Strain gauge)
eCAN	Enhanced controller area network
FFT	Fast Fourier transformation
FIFO	First in first out
FSB	Fail-safe bus
GPIO	General purpose input output
HMI	Human-machine interface
ISR	Interrupt service routine
JTAG	Joint test action group
MB	Main-bus
MK	Message-Kopf (Message head/tool)
N/A	Not available
PID	Proportional-integral-derivative
PW	Power
RX	Receiver
SCI	Serial communication interface
TX	Transmitter
UART	Universal asynchronous receiver-transmitter
XINTF	External interface