Control and implementation of compliant actuators for exoskeletons

handed in
FORSCHUNGSPRAXIS

cand. ing. Daniel Bargmann

born on the 07.11.1989
living in:
Moerikestrasse 1
83022 Rosenheim
Tel.: 0152 - 21966158

Chair of
AUTOMATIC CONTROL ENGINEERING

HUMAN ROBOT INTERACTION GROUP
Technical University of Munich

Prof. Dongheui Lee, Ph.D.

Supervisor:	Matteo Saveriano
Start:	02.11.2016
Delivery:	16.02.2017

Contents

Chapter 1

Problem Description

1.1 Mechanical Hardware

1.1.1 Series Elastic Actuators

Series-Elastic Actuators (SEAs) are actuators that incorporate a elastic element in series of the transmission, usually a spring (see Figure 1.1) inbetween motor and load. This results in a reduced bandwith for stable position control and a harder control problem in the position domain.
On the other hand the spring turns the force control problem into a position control problem, which makes it easier to achieve a stable force controlled system.
Electric motors in combination with traditional actuators have to depend on gear reduction to achieve sufficient acceleration for heavy loads. But gears introduce all kind of new problems, e.g.increase in weight and easy breakdown due to external impacts as those get amplified by the gear system.[PW]
One particular interesting feature of SEAs is its inherent compliance, which makes it a well suited approach for Exoskeletons. Here the human arm provides high damping so unstable behaviour doesnt occur that much. Most applications in industrial Exoskeletons also require to control the forces and typically position is not as important, as this is established by the human himself.

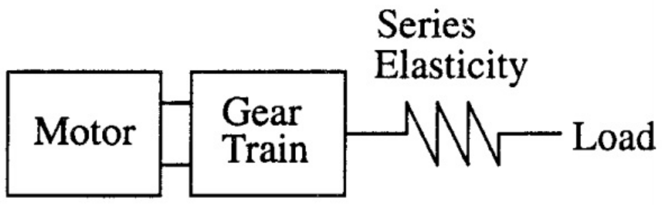

Figure 1.1: Concept of SEAs (from [PW])

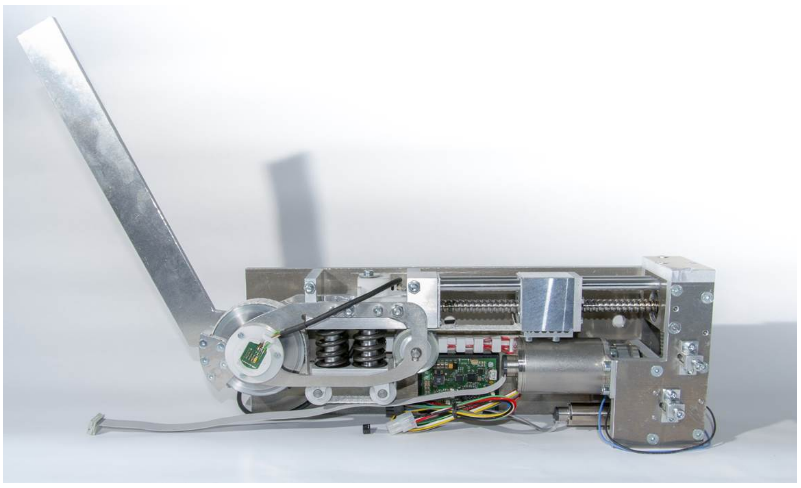

Figure 1.2: Orthosis Prototype version 1

1.1.2 Orthosis

The mechanical framework (Orthosis) is given by a SEA, a mechanical spring that is effectively in series with a DC Motor.
Our power orthosis consists of a joint, two springs in parallel, a spindle and nut, as well as two DC Motors.
The force of the motors is transmitted via a beltdrive which operates a spindle and nut. This spindle in turn transmits force via a belt to the orthosis joint where is passes two springs in parallel. Those springs effectively act as a series elastic element, which you can see in Figure 2.5.
The advantage compared to traditional springs in series is that the springs dont change their position when the orthosis is operated which makes it easier to measure the spring deflection with a displacement sensor. In order for the orthosis to be more

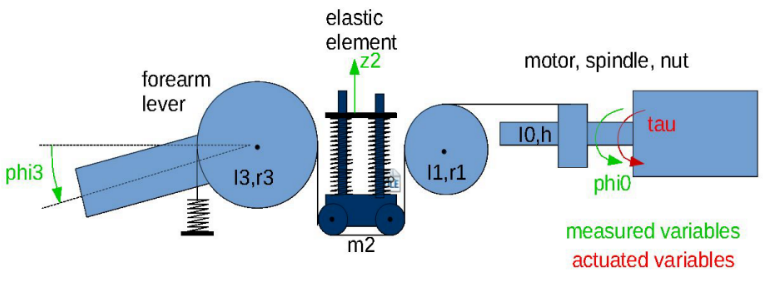

Figure 1.3: Schematic View of Orthosis

lightweight it is only able to support the user in the upward/lifting direction, so there is no need for an additional belt to operate the orthosis in the downward direction. Another speciality in our prototype is the use of two different DC motors.

The Maxon DCX 16 L is a traditional brushed DC motor with a nominal power of 19W which is intended to overcome the orthosis inherent inertia in a non-support or transparent mode. Therefor the nominal torque of 11.7 mNm of this motor is very tightly matched to the mechanical construction of the orthosis. It can be achieve a fast nominal speed of 10800 RPM which is needed to achieve a smooth operation where the user does not feel obstructed.

The Maxon EC-i 40 is a brushless DC (BLDC) motor with nominal power of 100W. This motor is intended to handle heavy lifting operations with its nominal torque of 3260 mNm albeit with a greatly reduced operating speed (nominal speed 3990 RPM).

In order to combine both motors both operate a belt drive which drives the spindle. In order to dynamically switch from transparent mode to support mode, the BLDC motor is connected to the belt drive via a electromagnetic clutch. If the orthosis should be operated in support mode the BLDC motor is clutched on and provides sufficient power. In the transparent mode on the other hand, the BLDC motor is not clutched on, so the smaller DC motor is able to react fast enough to the users movements.

1.2 Electronics

Certain electronics modules already existed and was developed and used in several previous projects at Fraunhofer IPA, especially the MYO Robotics Project[1]

Angular Sensor

Figure 1.4: Analog angular sensor RMB20

We used the RMB20 Angular sensor which provides an analog output of 0 to 5V for full 360° rotations. (Figure 1.4)

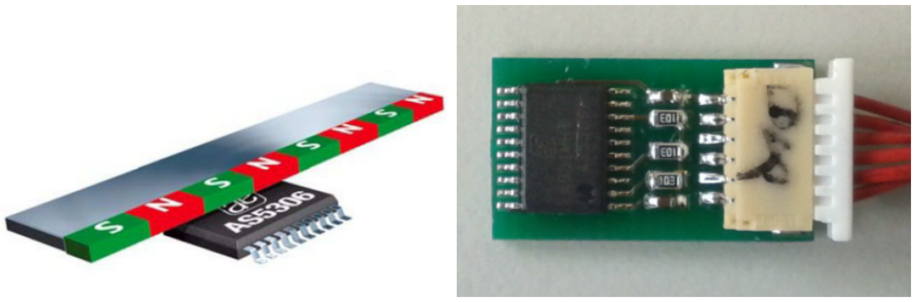

Figure 1.5: Left: basic principle of the displacement sensor. Right: PCB Board with displacement sensor

Displacement Sensor

This sensor is basically a incremental hall-sensor that works by sensing changes on a magnet strip. In our case one pole pair on the magnetic stripe corresponds to 2.4mm on the strip. Traveling along one pole pair results in 40 pulses of the sensor.(See Figure 1.5)
Therefore the maximum resolution is $\frac{2.4mm}{40counts} = 0.06\frac{\mu m}{count}$. In our case the encoder is configured in 4x mode, which results in a resultion of $15\frac{\mu m}{count}$.

Motorboard

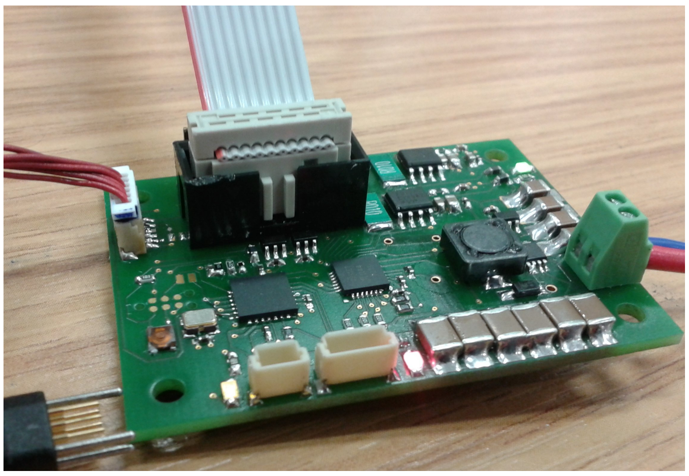

Figure 1.6: MYO Robotics Motorboard

This motorboard was originally designed for the MYO Robotics Project[1]. It features a driver for a BLDC motor and encoder counter for a incremental rotary encoder and one for the displacement sensor. Other sensordata are drawn current,

[1]www.myorobotics.eu

It can recieve data based on a certain protocol via a CAN Bus interface as well as via a SPI interface.
The motorboard an recieve a reference value via SPI which can be interpreted as either a PWM value or a velocity reference. It already has a simple PID controller implemented on its microcontroller, which regulates the velocity based on a given reference value.
It is designed for a power supply of 24V, which is converted to 5V via a buck converter and further to 3V to supply the internal microcontroller and communication ICs.

1.3 Task

The first part of the task is to establish a communication from a Linux PC to the orthosis and establish force control on it.
Therefore adequate hardware has to be found to connect the motorboard to a PC or even hardware to replace the motorboard.
In order to provide an interface to this framework for project partners and other projects, the implementation should feature a bridge to Robot Operating System (ROS).
After that the control should be optimized, its performance should be measured and the existing MATLAB model should be verified.

Chapter 2

Control Hardware Selection

2.1 Evaluation of different Hardware approaches

2.1.1 Microcontrollers

One way establish a control framework is by using embedded systems either as proxies for higher level controllers on other platforms (e.g. a PC) or containing the control algorithms themself.
We have evalued a few approches that feature embedded microcontrollers.

Synapticon SOMANET Modules

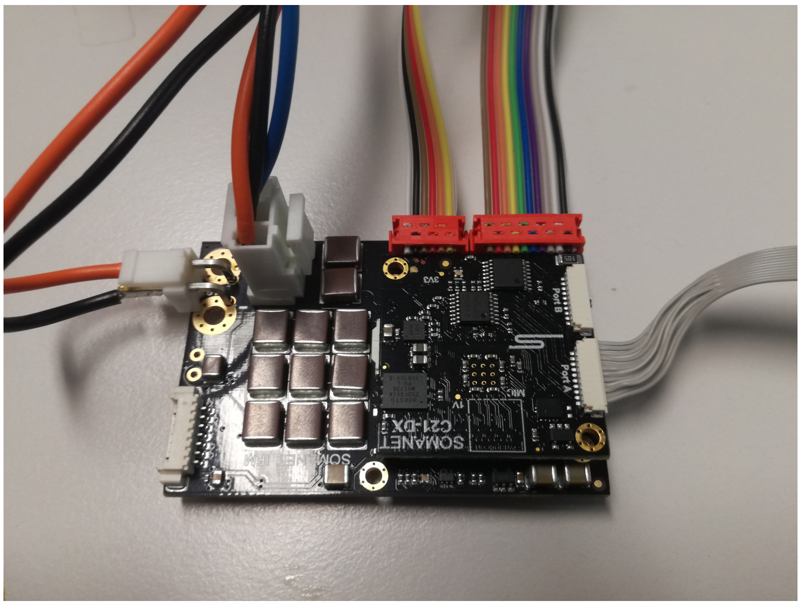

Figure 2.1: Synapticon SOMANET Module

Synapticons SOMANET Modules combine motordriver boards for BLDC motors with a computational core and communication modules like EtherCAT and SPI.
The compotational core features a highly parallel *XMOS* microprocessor which can spawn 16 deterministic independent real-time hardware threads while still beeing comparably low power (500 mW under load).[Syn14]
Another advantage are the communication moudles, like the ready-made EtherCAT slave module, modules for analog and digital general purpose input/output (GPIO) pins.
Unfortunatly it uses the programming language *XC*, a special derivative of C specially designed for *XMOS* microprocessors.
This makes it harder to write productive code for those microcontrollers as it requires learning the pecularities of *XC* as well as developing on embedded processors.
Also when using the *SOMANET* it is not possible (or even sensible) to use the already existing motorboard from the MYO Robotics project as the *SOMANET* core only works in conjunction with its BLDC interface.
This would require a complete rewrite of the code developed on the MYO Robotics motorboard, which is only sparsely documented.

Arduino YUN

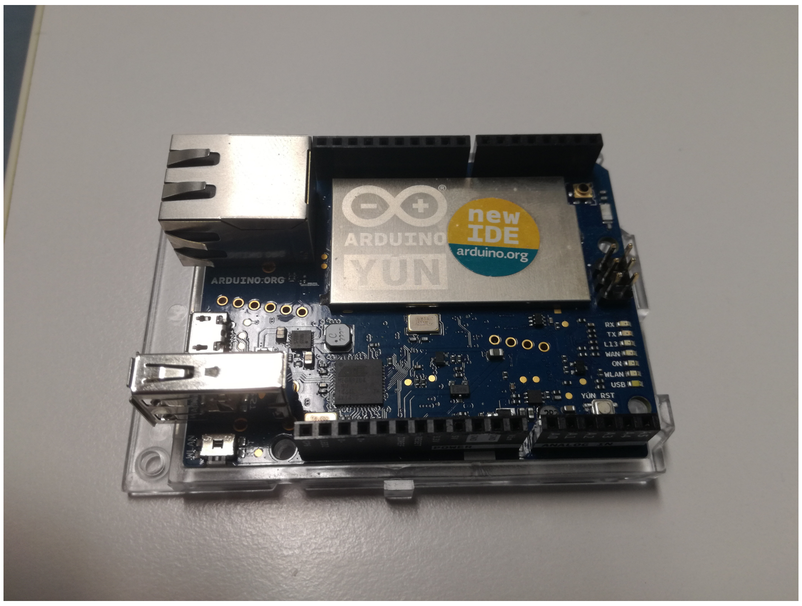

Figure 2.2: Arduino Yun

The YUN is an Arduino that incorporates a *ATmega32u4* microcontroller and a System-on-a-Chip (SoC), the *Atheros AR9331* on one PCB.
The Atheros processor runs the Linux distribution named *Linino OS* which makes implementing higher level controllers easier.

Unfortunatly the YUN only has one Serial Peripheral Interface (SPI) interface at the ICSP header, which either limits the amount of SPI devices that can be attached or the speed of the connection.
Connecting both displacement and rotational encoders is very tricky, as the *ATmega32u4* features two highspeed timers one for 64 MHz and one operating at 96 MHz. Each encoder needs two seperate timers to determine the revolution direction. So while it is technically possible to use the YUN to process encoders, the processing time can actually be severly limited if higher speeds are reached. One solution would be to use an external counter that is connected to the Arduino via Inter-Integrated Circuit (I2C) or SPI.

2.1.2 RoNex

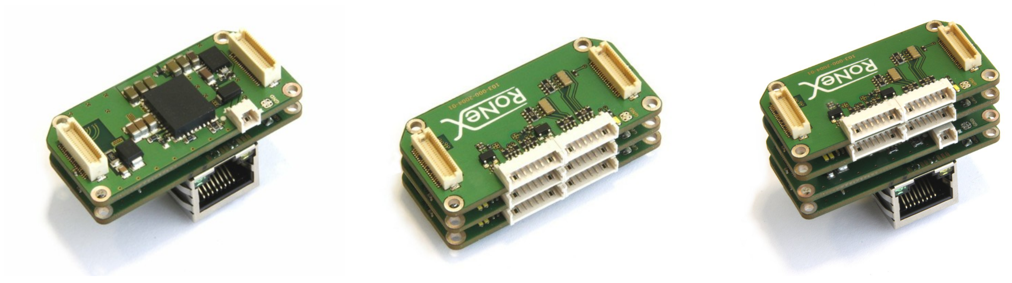

Figure 2.3: RoNex modules. Left: Base module — Middle: SPI module — Right: Both combined

The Robot Nervous System (RoNex) is a modular electronics hardware which was developed by *ShadowRobot*[1] to control their robotic hand.
It consists of several modules:

Bridge module connects other modules to a PC via EtherCAT, a protocol for real-time ethernet mainly established by Beckhoff[2]. Its driver has a hardcoded communication frequency of 1 kHz. It can be powered via Power over Ethernet and provides a power source for connected submodules.(see Figure 2.3)

SPI module connects to 4 SPI devices as Host and also features 6 digital input/output pins as well as 6 analog pins. It can be directly stacked on top of the RoNex base module and is powered by it.

It already supports ROS control out-of-the-box and is a well-tested system. The RoNex modules also provide a stable, real-time capable solution to connect to a motorboard and several sensors.

[1]http://www.shadowrobot.com
[2]www.ethercat.org

One downside is the low connection speed for SPI devices of only 32 bytes per connection and EtherCAT package, which is sufficient for our purposes.

2.2 Conclusion

For the final implementation we concluded that a complete rewrite of the motor-board (which would be required when using a microcontroller solution) would be too timeconsuming and prone to errors.
Instead using the RoNex board with its already well-tested hardware and relative ease of use is the best solution of all the proposed approaches.
This makes it also possible to use the PID velocity controller and the displacement sensor decoder on the motorboard.

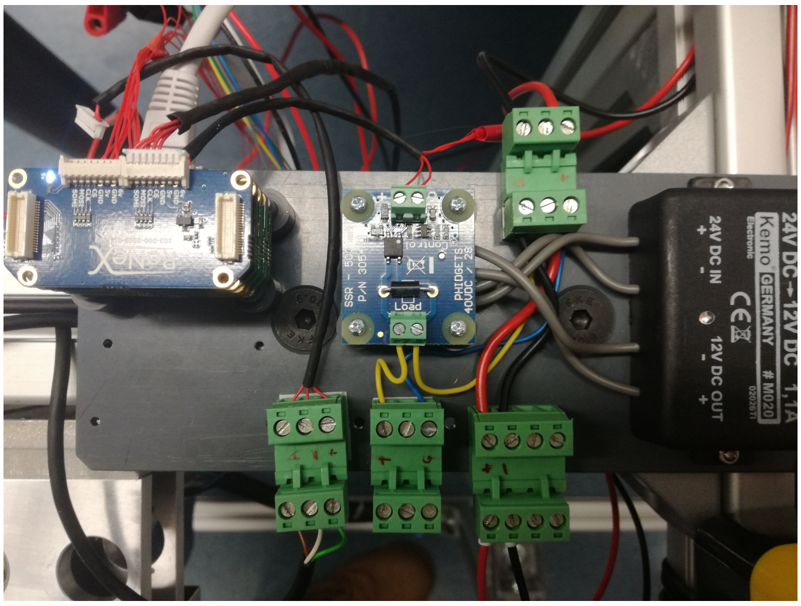

Figure 2.4: Final Implementation of the electronics

Ultimately we established the connection via RoNex while using a relay board for controlling the clutch (Figure 2.4)
The final schematic for the implementation can be seen in figure 2.5.

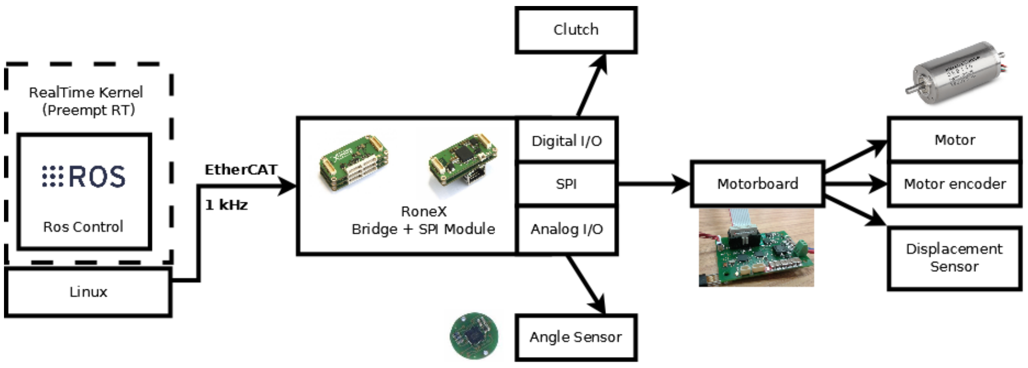

Figure 2.5: Final Implementation of the electronics - Schematic

Chapter 3

Control Software Selection

3.1 Realtime on Linux

The standard Linux kernel usually experiences a jitter of $\sim 1 - 10ms$ for a periodic task but for individual cycles jitter can be as high as several seconds.
When using Linux for motorcontrol it is therefore required to minimize the jitter so the calculated motorcontrol output matches the provided sensordata.
The main causes for high jitter are usually these [Hua14]:

Dynamic memory allocation and cache misses are one of the major impacts on high response latency. Whenever a cache miss happens the calculation is delayed by the time of a slow cache lookup.

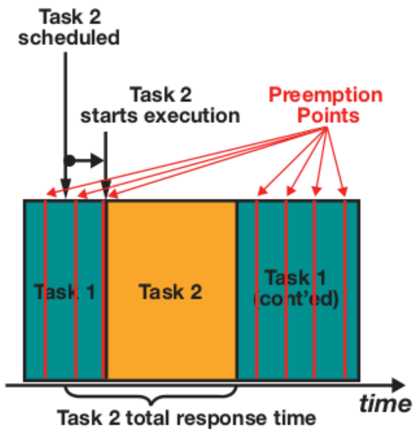

Figure 3.1: Preemptable Scheduler: Low priority Task 1 is paused as soon as high priority Task 2 gets scheduled and the next preempt point is reached

Non-preemptable Scheduler If a low priority task gets executed an an high priority task gets scheduled, the low priority tasks blocks its execution until it finishes.

It is very important for the Scheduler to be able to preempt low priority tasks and schedule high priority tasks. (see Figure 3.1)

Those problems can usually be addressed by using a realtime Linux kernel, which will be covered in section 3.1.2. This chapter is based on [RG16], [Hua14],[GMGT14]

3.1.1 Soft vs Hard Realtime

The main difference between Hard and Soft real-time is the way it handles deadline misses.

Soft real-time requires that the deadline is kept most of the time but doesnt count missing the deadline as failure. This is mainly used in Quality-of-Service (QoS) applications such as playing music or streaming videos. Here a missed deadline is not critical and the processed data (e.g. audio) is not useless or harmful when processing takes longer.

Hard real-time requires that the deadline is met 100% as missing it or using outdated information leads to malfunctioning of the system. This is usually important in safety-critical applications such as satelites or controllers with very tight stability margins.

The requirements for the power-orthosis lie somewhere in between hard and soft real-time. It is not critical to meet every single deadline especially if the velocity control is later on implemented on the motorboard.
On the other hand it is very important to decrease jitter for the control loop as much as possible as the frequency of the velocity reference usually has a high impact on control performance. Increasing the control frequency means in turn also increasing the impact of jitter on the performance.

3.1.2 Realtime Kernels

Various patches and kernel try to make the Linux kernel itself more real-time able. The two mainly used real-time kernels are *XenoMai* and *PreemptRT* which will be described in the following section.

XenoMai

Instead of improving the Linux kernel per se, Xenomai tries to inject a Real-time Operating-System (RTOS) in parallel to the traditional Linux kernel. This RTOS contains a seperate scheduler and has prefered access to system ressources.
Interrupt Requests (IRQs) get piped through *Adeos*, which handles IRQs and passes them to the RTOS kernel.(Figure 3.2)

[1]Adapted from https://xenomai.org/2014/06/life-with-adeos/

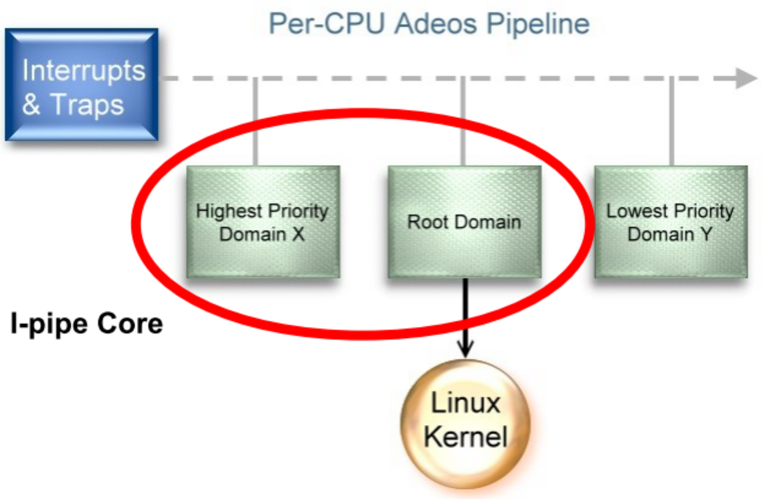

Figure 3.2: *Adeos* Pipeline[1]

PreemptRT

While XenoMai tries to circumvent the original Linux schedulier, PreemptRT approaches the problem differently. It patches the original kernel and modifies its scheduler to make the OS as preemptible as possible.
This means usually that interrupts need to be modified, e.g. by makeing them schedulable and/or disabling them

3.2 ROS and Realtime

ROS is a set of software libraries and tools for building robot application. It provides a defined and modular framework for organizing robot control programs even across several physical computers
It uses a protocol based on TCP/IP to communicate between *ROS nodes* which makes it inherently non-real-time.
To circumvent this limitation and still use ROS with real-time control several extensions can be used.

3.2.1 OROCOS

Open Robot Control Software (OROCOS) Toolchain establishes a real-time friendly and modular framework designed to create control system on Linux. It can be very tightly integrated with ROS and is often used in instances where controller design and performance is crucial for a robot. It can also handle different scheduler policies like earliest deadline fist or TDMA.

3.2.2 ROS Control

ROS Control is the native solution for controllers that have to fulfill real-time contraints in ROS. It was originally developed for PR2[2] Robots and was completely rewritten to be usable Controllers in ROS Control are usually very monolitic, as they consist of single plugins that get loaded by a controller manager which in turn runs them. Usually it provides only one policy for scheduling, first-in first-out.

3.3 Conclusion

The best solution when not considering the hardware setup would be to use XenoMai and OROCOS. XenoMai provides the best real-time performance and OROCOS is modular, features different scheduler policies, has many modules for dataprocessing already implemented and can be deployed to microcontrollers and is naturally supports XenoMai.
Unfortunatly it is quite a big amount of work to establish this framework for RoNex. The EtherCAT protocol is not specified in detail and although the source code is freely available, it would still be very time consuming to completely rewrite the already existing code for OROCOS.
Therefore we decided to use ROS Control. Unfortunatly the driver provided by *ShadowRobot* for now only supports PreemptRT and would require an adaptation to use XenoMai. Therefore we also use PreemptRT as real-time kernel.

[2]http://www.willowgarage.com/pages/pr2/overview

Chapter 4

Control

One of the major tasks is to implement a structure to control the Orthosis. This is basically a problem that revolves about controlling a SEA, but comes with a few additional problems.
The problem of controlling a SEA has been covered in a few different ways.
Originally [PW] proposed a *PID*-like control scheme based only on force-error and feedforward terms.
In [PW] and [Wye06], stability of *PID*-controllers has been shown for the actuators alone, but not in combination with an active or passive environment with deliberate impedance.
A popular approach to handle unknown environments is to utilize passivity based controller design, as a passive system is stable and a combination of passive systems again is passive.
To achieve this, [VEVDKB07] and [Wye06] used a cascaded control with an inner velocity loop to ensure passivity, where passivity can be ensured by correctly choosing controller parameters.
A major problem however remains that the classical approach based on *PID*-Controllers slows down the response of the system. The integral part of the controller basically acts as a low-pass filter [PW].
In order to counter this problem, [CCF14] evaluated integral sliding mode Control and an Adaptive Force Controller approaches and compared it to traditional *PID* control.
They concluded that basically adaptive and sliding mode control can achieve stability in sense of boundedness while at the same time achieving superiour tracking performance to *PID/PD* control.
In [JB16], a very similar mechanism to ours has been discussed. A motor drives a torsional spring via a pulley. This torsional spring is connected to the exoskeleton frame. So the interaction force can be calculated by comparing the the measurements of motor position and exoskeleton joint position. They tried to control the structure by using a combination of *PD* controller, Disturbance Observer (DOB) and a Zero Phase Error Tracking (ZPET) feedforward element. Our implementation is similar,

as we also transform the force control problem to a position problem by linking the force on the spring to its displacement.
Their results show that a PD controller can achieve quite good performance while at the same time a DOB doesn't provide much improvement. A ZPET Feedforward element on the other hand improves the performance quite dramatically. It isnt shown on the other hand if the performance improvement is limited solely on the ZPET element or if it is the effect of a Feedforward element in general.
All in all we conclude that a cascaded *PD*-Controller should be sufficient for our purposes as first prototype. In contrast to [VEVDKB07], which also utilized a cascaded controller, we refrain from using an integral part.
It is also apparent that the spring already introduces a integral factor into the system.
The spring is defined by

$$F_{support} = F_{spring} = kz \tag{4.1}$$

where k is the spring constant and z is the displacement. With $sZ = sX = V$ we can conclude that

$$F_{support} = \frac{1}{s}kV \tag{4.2}$$

This is obviously a very simplified model and does not describe all of the system in the given orthosis, but it shows that the plant already includes an integrating factor.
Therefore we conclude that it is not necessary to include an integrating factor into the controller.
This is because we think it is unnecessary to exactly track a certain force, as it should act as support and is guided by a human. Therefore an exact tracking is not important, but fast response is.
On the other hand, due to the fact that the provided model of the orthosis doesnt match real-world data very well, it is unfortunately apparent that the model isnt sufficient for controlling the orthosis.
Another problem is that due to mechanical problems, where we had to remove the small motor and fixate the clutch, the system properties changed quite a bit.
Because more advanced methods of control, such as proposed above require very detailed modelling beforehand, this could not be achieved due to time constraints.
The first tested architecture can be seen in figure 4.1.
It features a *PD*-Controller for a inner motor velocity control loop, a simple *P*-Controller for the outer control loop that controls the displacement and a feed-forward element for the velocity loop, which mainly counters stiction.

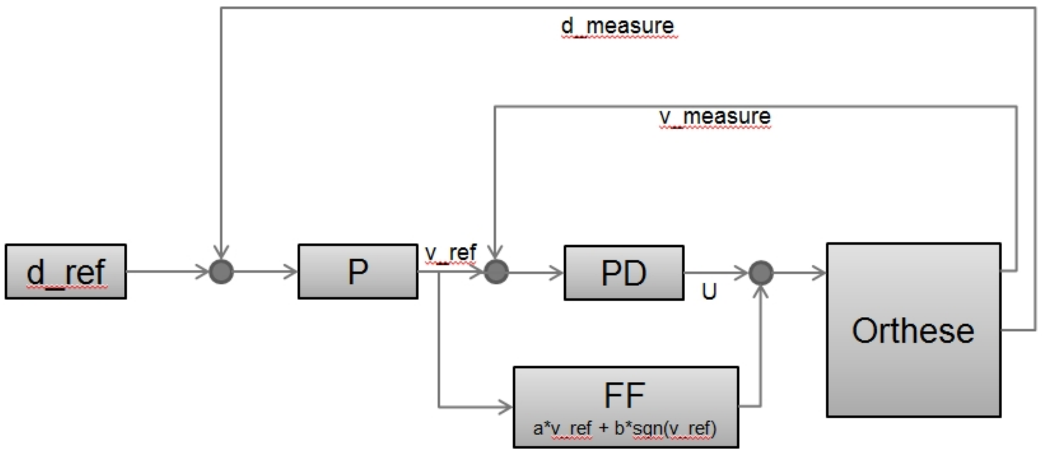

Figure 4.1: First tested control architecture

Chapter 5

Implementation

First, there arose quite a few problems with the mechanics and electronics.

5.1 SPI Interface

One major problem was the implementation of the SPI protocol. As it was really insufficiently documented, the first tries of getting the motorboard to work did not work.
We didnt get any sensordata from the motorboard and we couldnt get the motor to spin with SPI interface, although with the Controller Area Network (CAN)-Interface it was certainly possible. Usage of the CAN-Interface wasnt recommended though, as it is implemented with a very slow control frequency of 5 ms (at least) and was originally only intended for debugging purposes.
As the SPI-Interface hasnt been used before, we evaluated the protocol with an oscilloscope in found out that the protocol had a small error, where the order of two values was interchanged.
After considering that we could spin-up the motor and get sensible sensordata.
To make it usable in the long run, we programmed a C++ library for Arduino, a FTDI FT4222H controller device and the Kvasar CAN Interface, which we use to communicate via CAN, respectively, in order to make it easier for others to work with the motorboard.

5.2 Implementation with RoNex

The implementation with RoNex required existential rework of the driver.
To access the RoNex device's analog, SPI and digital ports, originally it is intended to use ROS directly.
Because ROS is not real-time compatible, this is not a sufficient solution in our case, as we require real-time able communication.

Therefore, the real-time able part of the driver has to be harnessed, which in essence is based on ROS-control.
The driver basically has a seperate loop that takes care of the EtherCAT communication. Here, data from and to the RoNex gets packed and unpacked from EtherCAT packages. (see figure 5.1)
In order to provide input and output in a real-time compatible fashion, a controller manager - part of the ROS-Control framework - takes care of spawning and calling the individual controllers.
The driver takes data from the EtherCAT package and places it into the interfaces for the controllers and also places data back into a buffer that gets send back to RoNex in an EtherCAT loop.
It then calls the controller manager which in turn calls the controllers which calculate the next control output.
Unfortunatly, the SPI module was not intended to transfer the analog and digital data to the controller directly, but only pass it by via a ROS topic which was initialized in the main driver.
In order to access this data in a real-time safe environment, we needed to introduce a new interface for the controllers, which had to inherit from an already existing class named CustomHW.
This proved to be very time consuming in the end, as no extensive documentation was available for the driver.

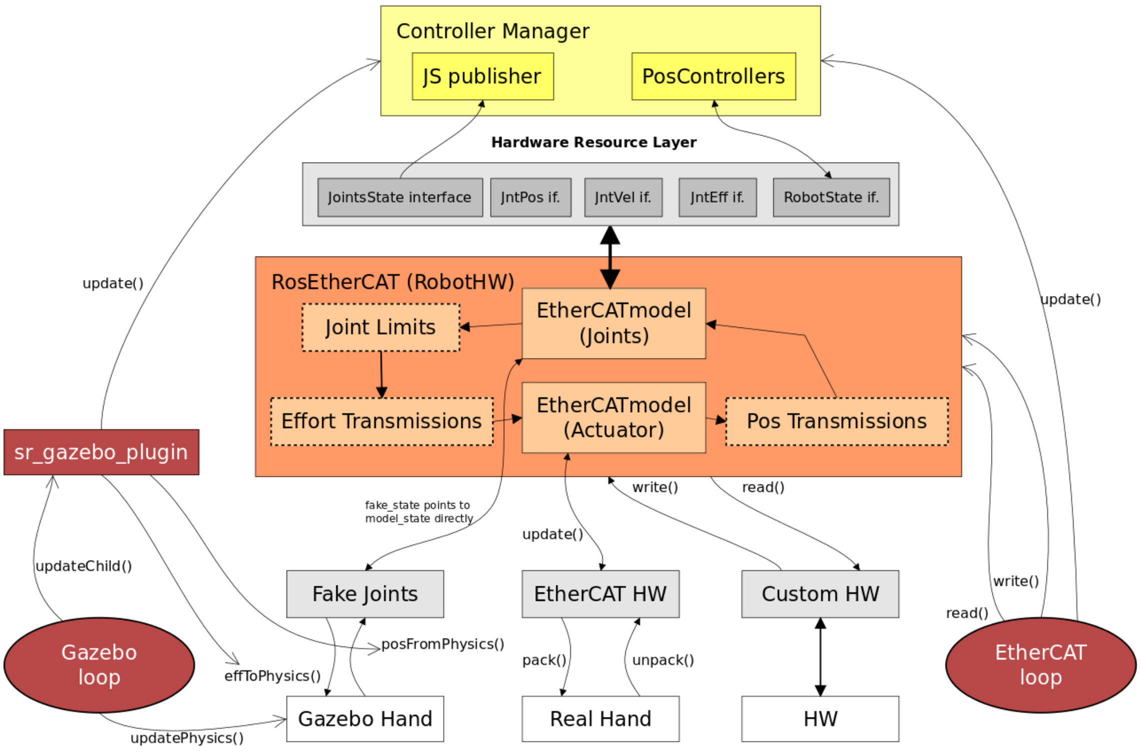

Figure 5.1: ROS RoNex connection[1]

5.3 Motorboard bug

One of the main issues that happend was the complete outage of the motoboard. In order to evaluate this, we tested a few issues.
The bug generally occures when changing the velocity reference - and therefore the given voltage - too rapidly.

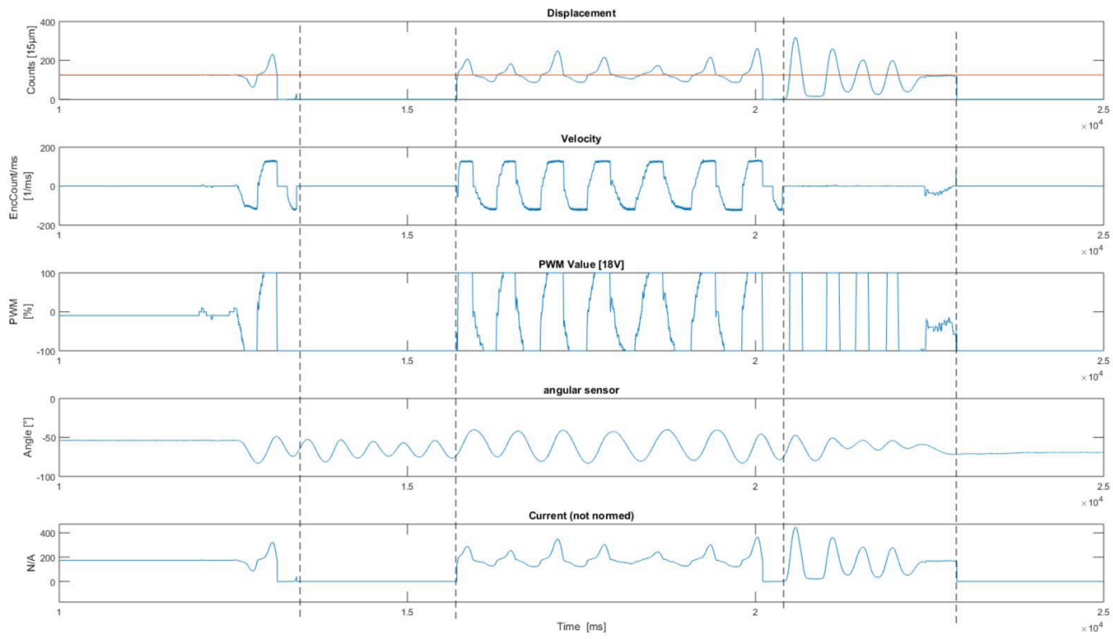

Figure 5.2: Motorboard bug total

As one can see in figure 5.2 at a certain point (ca. 10s) all sensor values provided by the motorboard (velocity and spring displacement) are reset to -1, which is the error code returned by the controller in case there was no SPI-message from the motorboard.
Other sensorvalues like the angular sensor - an analog sensor which provides its values via the RoneX analog port - provides accurate and sensible data.
This total loss of all sensor data is the first kind of bug. The second kind is observed later on at about 21s.
Here sensor data is not completely lost. In this case the displacement sensor still yields sensible data and the calculated PWM value still varies but it does not result in the appropriate velocity, as one can see from the velocity sensordata.
In both cases the motor stalls but in the rst case it doesnt move at all whereas in the latter it still executes (very) small movements (as can be seen better in figure 5.3).

[1]From Shadow Robots Github page : *https://github.com/shadow-robot/rosethercat/*

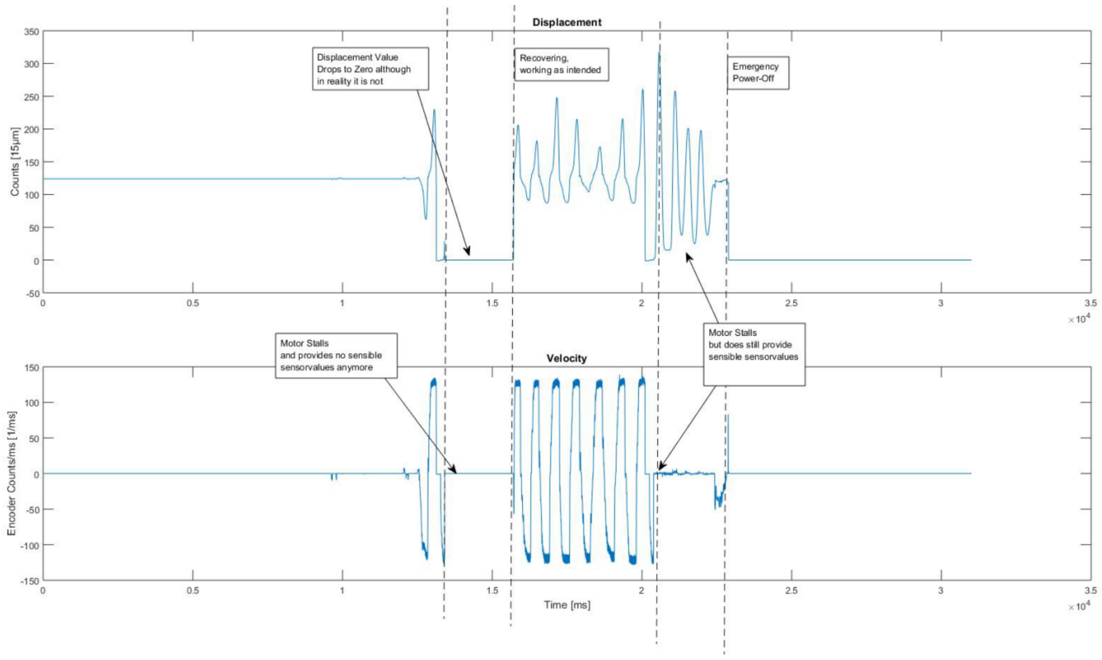

Figure 5.3: Motorboard bug

Afterwards another scenario is tested, where the big motor is decoupled from the main part of the orthosis in order to evaluate the bug further.

The motor was operated at 37.5/100 PWM and then the given PWM was sud- denly changed to -37.5/100.

Afterwards the motor accelerated for a very brief period and then stopped completely while the motorboard provided no sensor values at all. After a few seconds the motor accelerated again, kept a certain velocity and returned sensible sensor data.

Meanwhile the input voltage accross the motorboard was measured. As seen in figure 5.5 a peak in voltage of over 25V up to 28V occurs for a very short time (approximately 10 us, see figure 5.4).

Apparently the supply voltage experiences a sudden increase of about 3V when the bug happens. This is probably caused by a security shutdown of the step-down converter when it hits 25V, which is the absolute maximum rating of the converter. This removes the load caused by ICs on the 5V and 3.3V lines.

In turn this leads to a sharper increase of voltage to about 28V (peak) for 10 us. This overshoot then gets regulated by the voltage supply (EA-PS3065-10B), but its reaction is too slow (it regulates from 90-10 % load in less then 3 ms)

Apparently the malfunction of the step-down converter leads to a further increase in voltage, which increases the chances of permanently damaging the step-down converter.

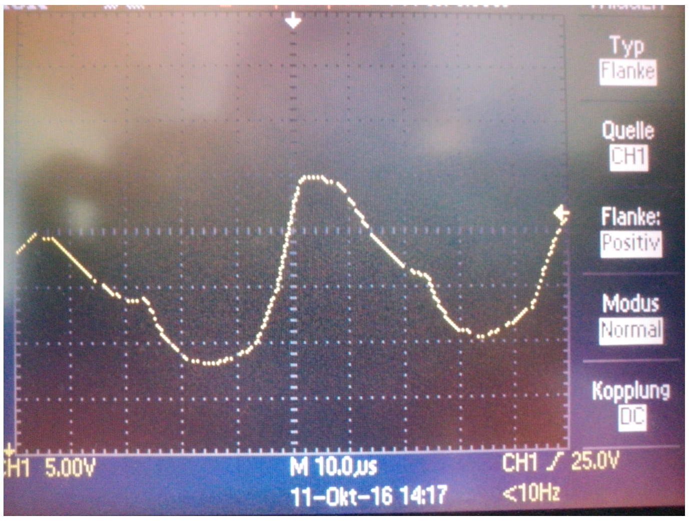

Figure 5.4: Motorboard bug

5.4 Mechanical Problems

There occured several other problems. One major set of problems was related to the mechanical design of the clutch.
In one of the first tests, the clutch exhibited a loud noise, which we thought was dangerous.
After a bit of research, this bug was found to be the result of non-centrical alignment of the two sides of the clutch. This resulted in a periodical increase in friction between the two clutch plates even when the clutch was decoupled.
Because the problem was mainly the result of insufficient mounting of the motor and clutch (they moved when they were actuated), to deal with this, a hull for the clutch was designed to provide additional support for the motor.
Another problem which also arose was similar to the last one. In a test run the small motor overheated as it also wasnt mounted sufficiently and moved.
This caused additional stress on the motor and resulted in an overheating of the motor.
As a result we decided redesign the mounting of the small motor as well as using a more powerful BLDC motor. Changing the motor was mainly due to the fact that we can use the same motorboard for both motors in doing so and also because friction is much higher than anticipated in the first calculations, so a more powerful motor is mandatory.

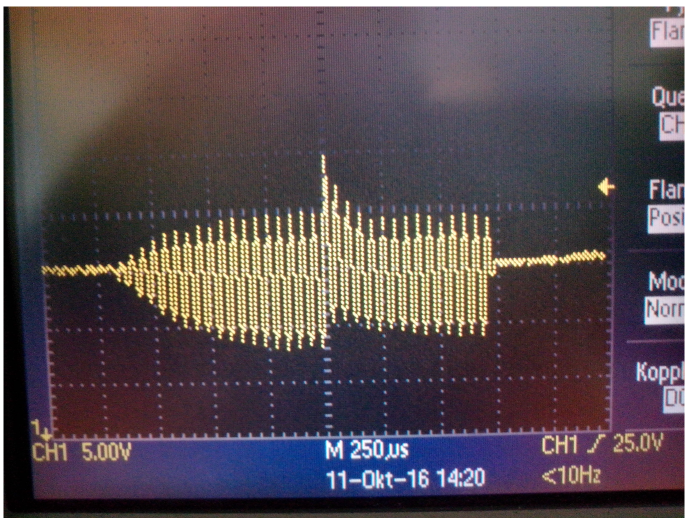

Figure 5.5: Motorboard bug

Chapter 6

Evaluation & Performance

First an few already by another student developed controllers were tested

A cascaded double PI controller (same structure as 4.1) with values of $P = 10$ and $I = 3000$ for the outer loop and $P = 30$ and $I = 3$ for the inner loop was tested. This controller proofed to be unstable.

A cascaded double PD controller (same structure as 4.1) with values $P = 20$ and $D = 5$ for the outer loop and $P = 10$ and $D = 3$ for the inner loop was tested. This controller proofed not to be stable.

After those two tests we concluded that the previous developed model of the orthosis was not sufficient. We additionally concluded that before further evaluating the model, we would wait for the next iteration of the orthosis to be build, which will also feature a spring to counter the difficult to control non-linearity introduced by the one-sided transmission of force through the belt.
Because of this we postphoned correcting the model until the next iteration of the orthosis is available.
To still provide a proof of concept of the the orthosis and to set it at motion, we designed a motor controller with traditional loop shaping.
We took the traditional widespread model of electrical motors

$$\frac{d}{dt}\begin{bmatrix}\dot{\theta}\\ i\end{bmatrix} = \begin{bmatrix}-\frac{b}{J} & \frac{K}{J}\\ -\frac{K}{L} & -\frac{R}{L}\end{bmatrix}\begin{bmatrix}\dot{\theta}\\ i\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{L}\end{bmatrix}V \tag{6.1}$$

whereby θ is the position of the motor, i is the current, K is the back EMF constant, J is rotor inertia, b is the friction constant and R and L are electric resistance and inductance respectively
We used this with the output $\dot{\theta}$ to calculate the transfer model and based on that we calculated the values for the *PD*-Controller with *MatLAB* via loop-shaping.
Therefore we inspected the sensitivity function S and its complementary counterpart T and designed the controller so that the roll-off frequency for T and S is at 10 kHz.(see figure 6.1) This resulted in the values of $P = 10$ and $D = 0.01$.

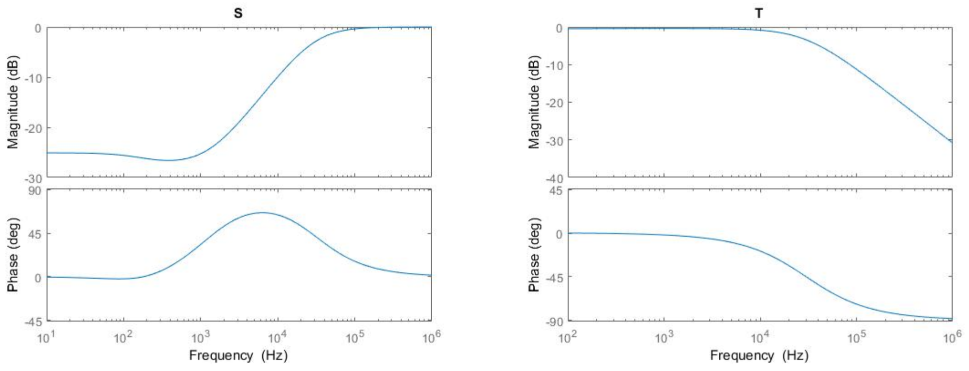

Figure 6.1: Bode plots of sensitivity function S and its complementary counterpart T

For the Feedforward element, we let the motor run from maximum to minimum duty cycle and measured the velocity that occured without any load. These measurements we matched to a model of $V = av_{ref} + bsign(v_{ref})$ with linear regression. The result you can see in figure 6.2. We used this as our feedforward element, which resulted in $a = 27$ and $b = 126$.

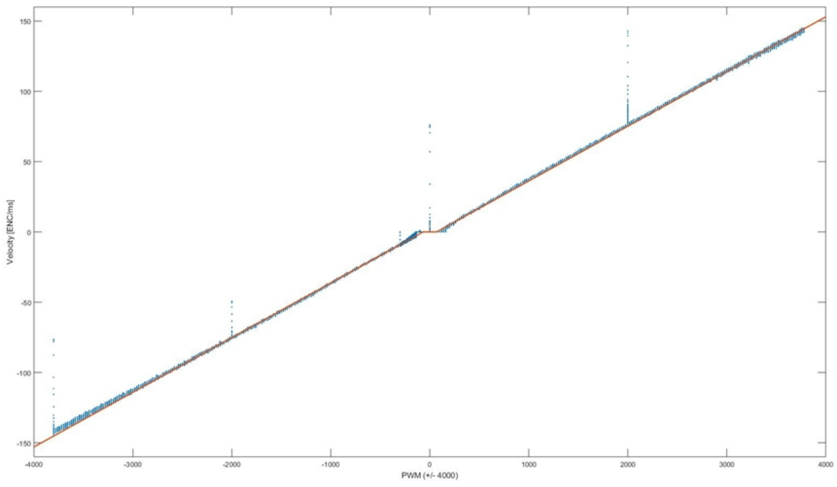

Figure 6.2: Measurements of several iterations of motor velocity to given PWM duty cycle and fitted regression model (red)

For the outer loop we used trial-and-error methods, which resulted in a stable mode for $P = 10$.
In a final test we evaluated the controllers stability in a test run, which you can see in part in figure 6.3. (ommitted part of the data in the plot to avoid clutter)

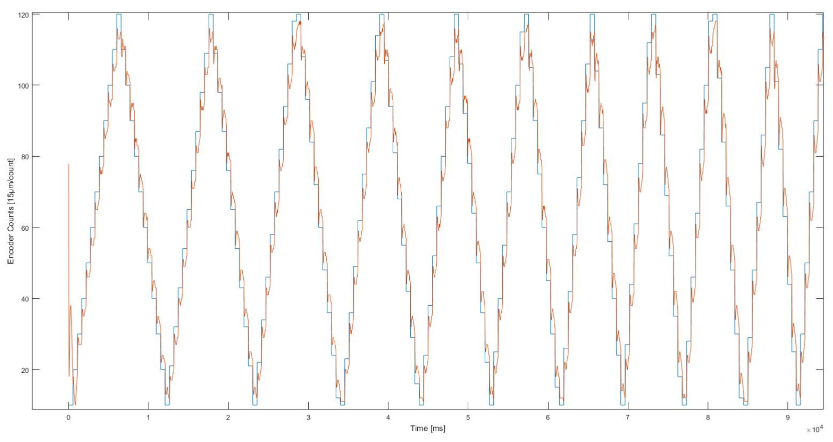

Figure 6.3: Part of the final testrun of the orthosis

Chapter 7

Conclusion & Outlook

The final test rig can be seen in figure 7.1 and 7.2.
We established a framework to control the orthosis and provided feedback for the mechanical design and improved on it. The test rig is functional and sufficiently performant.
Furthermore we evaluated a few state-of-the-art approaches to the problem of SEA based systems, where quite a few were used in conjunction with exoskeletons.
On this basis we suggested a few approaches for future implementation. The next step is to iterate on the mechanical design and incorporate the small motor into the control framework.
Based on the reiterated prototype the model should be revisited and errors in the model can be corrected based on that.

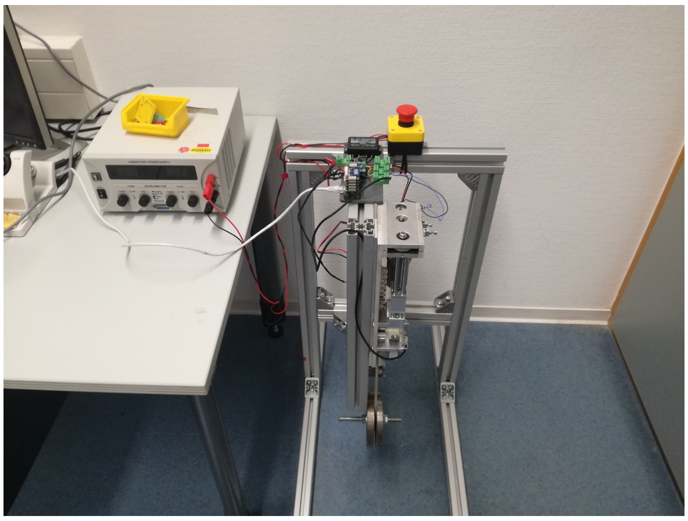

Figure 7.1: Orthosis test rig complete

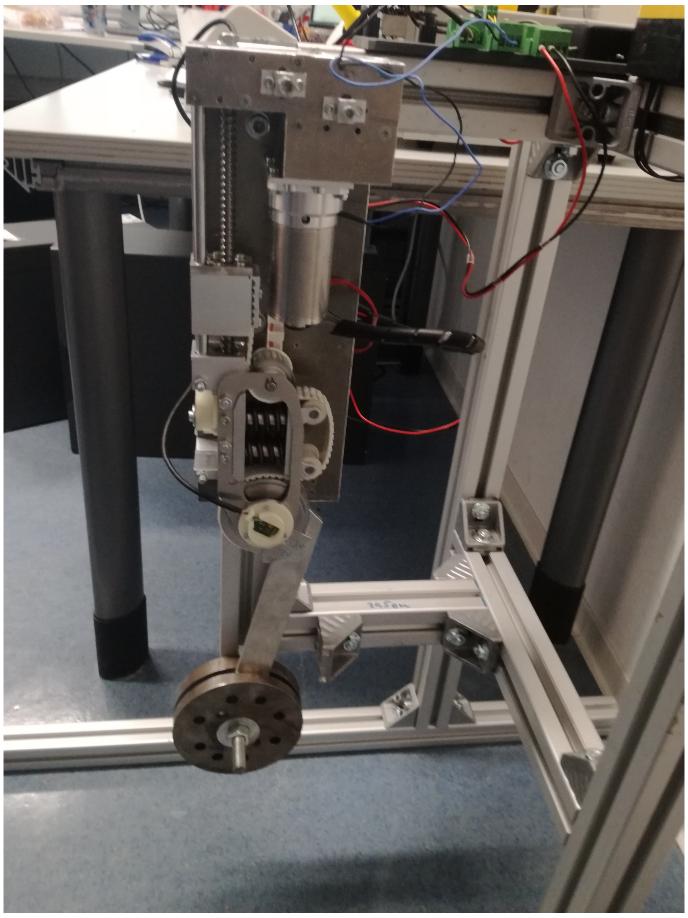
Figure 7.2: Orthosis side view

Acronyms and Notations

SEA Series-Elastic Actuator

BLDC brushless DC

GPIO general purpose input/output

SoC System-on-a-Chip

QoS Quality-of-Service

RTOS Real-time Operating-System

IRQ Interrupt Request

ROS Robot Operating System

OROCOS Open Robot Control Software

SPI Serial Peripheral Interface

RoNex Robot Nervous System

I2C Inter-Integrated Circuit

ZPET Zero Phase Error Tracking

DOB Disturbance Observer

CAN Controller Area Network

Bibliography

[CCF14] Andrea Calanca, Luca Capisani, and Paolo Fiorini. Robust force control of series elastic actuators. *Actuators*, 3(3):182–204, jul 2014. URL: `https://doi.org/10.3390%2Fact3030182`, `doi:10.3390/act3030182`.

[GMGT14] Carlos Garre, Domenico Mundo, Marco Gubitosa, and Alessandro Toso. Performance comparison of real-time and general-purpose operating systems in parallel physical simulation with high computational cost. In *SAE Technical Paper Series*. SAE International, apr 2014. URL: `http://dx.doi.org/10.4271/2014-01-0200`, `doi:10.4271/2014-01-0200`.

[Hua14] Jim Huang. Rtmux: A thin multiplexer to provide hard realtime application for linux. *Embedded Linux Conference Europe*, 2014.

[JB16] Yeongtae Jung and Joonbum Bae. An asymmetric cable-driven mechanism for force control of exoskeleton systems. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Institute of Electrical and Electronics Engineers (IEEE), oct 2016. URL: `https://doi.org/10.1109%2Firos.2016.7759066`, `doi:10.1109/iros.2016.7759066`.

[PW] G.A. Pratt and M.M. Williamson. Series elastic actuators. In *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*. Institute of Electrical and Electronics Engineers (IEEE). URL: `http://dx.doi.org/10.1109/IROS.1995.525827`, `doi:10.1109/iros.1995.525827`.

[RG16] K. Sripath Roy and K. Gowthami. Implementation of xenomai framework in GNU/linux environment to run applications in a real time environment. *Indian Journal of Science and Technology*, 9(17), may 2016. URL: `http://dx.doi.org/10.17485/ijst/2016/v9i17/93109`, `doi:10.17485/ijst/2016/v9i17/93109`.

[Syn14] Synapticon GmbH. *SOMANET Core C21 DX Datasheet*, 2014. Rev. A.5.

[VEVDKB07] Heike Vallery, Ralf Ekkelenkamp, Herman Van Der Kooij, and Martin Buss. Passive and accurate torque control of series elastic actuators. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 3534–3538. IEEE, 2007.

[Wye06] Gordon Wyeth. Control issues for velocity sourced series elastic actuators. In *Proceedings of the Australasian Conference on Robotics and Automation 2006*. Australian Robotics and Automation Association Inc, 2006.

License