Master's Thesis

Surface EMG Based Tuning of Bilateral Telepresence Systems

Albert Arquer Cortés

DLR

Robotics and Mechatronics Center	BJ.: 2013
	IB.Nr.:

MASTER'S THESIS

SURFACE EMG BASED TUNING OF BILATERAL TELEPRESENCE SYSTEMS

Author:

Albert Arquer Cortés, B.Eng _____

Supervisors:

Dr. Jordi Artigas Esclusa _____

Dr. Claudio Castellini _____

The Intitute's head

Prof. Dr. G. Hirzinger _____

This document comprises X pages, Y figures and Z tables

Oberpfaffenhofen	Date: March 2013	Author: Albert Arquer Cortés	Mark:

ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA
ELECTRÒNICA I INFORMÀTICA LA SALLE

TREBALL FINAL DE MÀSTER

MÀSTER EN ENGINYERIA ELECTRÒNICA I AUTOMÀTICA

SURFACE EMG BASED TUNING OF BILATERAL TELEPRESENCE SYSTEMS

ALBERT ARQUER CORTÉS JORDI AlBÓ CANALS

ACTA DE L'EXAMEN
DEL TREBALL FINAL DE MÀSTER

Reunit el Tribunal qualificador en el dia de la data, l'alumne

D. ALBERT ARQUER CORTÉS

va exposar el seu Treball Final de Màster, el qual va tractar sobre el tema següent:

SURFACE EMG BASED TUNING OF BILATERAL TELEPRESENCE SYSTEMS

Acabada l'exposició i contestades per part de l'alumne les objeccions formulades pels Srs. membres del tribunal, aquest valorà l'esmentat Treball amb la qualificació de

Barcelona,

VOCAL DEL TRIBUNAL VOCAL DEL TRIBUNAL

PRESIDENT DEL TRIBUNAL

Declaration

I certify that this thesis is entirely my own work. Where any external source of information has been used, it has been acknowledged. This work has not been submitted, either in whole or in part, for a degree at this or any other university.

Place	Date	Albert Arquer Cortés

Abstract

The controller in a bilateral teleoperation system is often design in order for the system to behave analogous to a light rigid bar. This is motivated by the fact that such a tool would provide a perfect feedback, both faithful and immediate. However, this approach has some limitations, specially when dealing with delayed feedback or systems in which the slave's motor skills are inferior to the operator's. In such cases stability constraints often entail a decrease in transparency.

This thesis presents an alternative method in which a position controller is dynamically tuned to mimic the dynamics of the operators arm. The controller can then be, at least to some extent, regarded as an *extension* of the operators limb. Electromyographic data was used in order to estimate the subject's limb stiffness. The relationship between this variables was then analyzed and, as expected, it was found to be very strong, leading to correlations of over 85% in some subjects. Finally some evaluations were conducted by following certain trajectories under low and high stiffness. The results certified the system's capacity at tuning the controller's stiffness and the expected results involving a controller with low stiffness and one with high stiffness were obtained.

Contents

List of Figures

List of Tables

1

Introduction

Telepresence could be described as the set of technologies that allow a person, called the *operator*, to feel as if they were present at a place other than their real physical location. The perfect telepresence setup is envisioned as one that would not allow the operator to tell the difference between directly interacting with the environment and doing so through the system, therefore, telepresence ideally involves the remote replication of all capacities and senses.

Telepresence systems are currently being used in tasks where the direct presence of a human operator is not desirable, for example, those involving remote or harsh environments such as explosives disposal or nuclear plant maintenance tasks. It is believed that telepresence systems will gain importance in the future and either serve as a midpoint towards the development of autonomous robots or as a permanent solution to those tasks where this latter approach is not feasible or efficient.

An area of paramount importance to the field of telepresence is that concerned with the sensing and replication of the operator's motor capacities such as touch and force. This field, known as *haptics*, has received a lot of attention since the concept of telepresence was first introduced in 1980 [13] and is in fact the topic with which this thesis will be concerned. The means of implementing such features has evolved through many stages, from the usage of a simple rigid rod linking master and slave to complex arrangements of electromechanical actuators, force and torque sensors, and communication channels. These complex entities obviously require a lot of attention to ensure the correct behavior of a telepresence system, for example, actuators need to be closely controlled to ensure the safety of the operator as this devices are potentially harmful, also, the communication

channel used requires rigorous analysis since, as it will be discussed later, any delay can easily render the system unstable.

A surgical telepresence system currently in use at numerous hospitals can be seen in Figure 1.1.

Figure 1.1: The da Vinci® Surgical System developed by Intuitive Surgical, currently deployed in several thousands hospitals worldwide, is a telepresence system designed to perform complex surgery by minimal invasion procedures. It is estimated more than 200.000 surgeries were performed with this system during 2012. ©2013 Intuitive Surgical, Inc.

1.1 Teleoperation with Force Feedback

As the term itself suggests, teleoperation is the ability to operate from a distance, so in a way, teleoperation can be seen as a part of telepresence. Although the term teleoperation can refer to many kinds of operation, our treatment of the subject will be limited to the most *physical* approach. In other words, the term teleoperation, or tele-manipulation for that matter, will be used to refer to the capacity of physically interacting with the remote environment.

Current teleoperation systems are generally based on sampling a series of variables from one end of the system, and after applying some kind of transformation to them (the entity in charge of such is commonly known as the *controller*), replicating them at the other side, either unilaterally, (the sampling and replication only takes place in one direction) or bilaterally (taking place in both directions). The most widely used physical variables are spatial positions and applied forces and torques. Force and torque sensors are based on measurements of elastic strain on specially designed structures, because of technical reasons, these sensors are expensive to manufacture and usually have a high offsets and gain variance, requiring individual calibration. Because of this, simple teleoperation architectures tend to rely on position measurements. While position measurements can be very useful and can be acquired by relatively simple means, they provide no data concerning

the dynamic interactions taking place between the master or slave and the operator or the environment, a problem that as it will be seen, impedance controllers attempt to fix [9] .

Another crucial characteristic of any teleoperation system is its capability to apply forces and torques back in the master's side of the system. Force feedback allows us to transfer the current operator's impedance through the system up to the slave side, this happens automatically in a system based on force measurement and through the use of a impedance controller with a high enough stiffness in a system based on position measurements. Any system lacking force feedback[1] is severely limited in the sense that it is unaware of the dynamic characteristics of the operator and therefore the interactions occurringing at the remote site of the system will be in accordance to a pre-stablished model.

As an example of the importance of force feedback the following example is presented. Imagine the operator of a teleoperation system without force feedback is trying to pick up an egg by closing a tweezor-like slave robot situated around it. Since the system has no force feedback, the operator has no way of knowing the interaction forces occurring between the slave and the environment, in other words, the operator does not know if the slave is either loosely or tightly locked around the egg. This lack of feedback makes it practically impossible for the operator to place the slave robot at the exact position required to perform the given task (in this case, picking up the egg without breaking it), and would cause the slave robot to either not come in contact with the egg at all, and therefore applying no force, or exerting the maximum available force around the egg while trying to reach the position of the master and therefore breaking the egg. The only solution in such scenario would be to incorporate a static impedance controller so that the force applied by the slave would be proportional to the deviation in the position with the master, still, the operator would have no direct feedback of force that is being applied to the egg shell, making the task very un-intuitive.

In conclusion, by any reasonable definition, tele-manipulation involves a physical interaction between the machine and the environment, it is here where force feedback plays a crucial role as it allows to replicate forces back at the operator's side. These forces can either be direct measurement taken at the other side of the system if it is equipped with force sensors or they can be computed based on displacement errors through an impedance controller.

[1] For the sake of brevity force and torque feedback will be referred as force feedback.

1.2 Electromyography and its Applications

Electromyography (EMG from now on) can be defined as any method of analysing and recording the electrical activity generated by muscles of the human body, or any other organism with similar locomotion means for that matter. To be able to understand a little better what EMG does measure and what it does not measure a basic understanding of the human motor system is required.

1.2.1 Human Muscles

The human body contains approximately 642 skeletal muscles. Fortunately enough, most of those are pretty much irrelevant to the motor capabilities of a human body limb. For example a human face contains a convoluted mesh of muscles which, as far as many activities go, walking just to mention one, are pretty much useless, since they are mainly used to modify facial expression.

A muscle is basically a bundle of protein filaments or *fibers* which have the capacity of sliding longitudinaly one on top of each other and therefore modifying both length and thickness of the whole tissue. It is through this elongation and contraction process that muscles manage to generate force and therefore motion. Although different kinds of muscles exist this thesis is going to be concerned with a certain type called *skeletal* muscles, the name steaming from the fact that their function is management of the skeletal position. Other kind of muscles would be *smooth* muscles and *cardiac* muscles which are in charge of involuntary or semi-involuntary functions like breathing or the contraction of heart so they have little or none impact on the overall skeletal dynamic state. While whole books can be and are written concerning the detailed anatomy and working principles of muscles it is way out of the scope of this thesis to go in-depth on this topic [11].

1.2.2 Volitional Muscle Control

All muscles in the human body are ultimately controlled by the nervous system. Although this control can involve different parts of this system, skeletal muscles, especialy the voluntary contraction and relaxation of these, generally share the same controlling scheme. The complex brain functions involved in the contraction and relaxation of a muscle will not be discussed, for our purposes we will only consider voluntary muscle movement and we will limit the analysis of the system starting at the neuron of the motor cortex (brain) which eventually sends the *signal* down the spinal cord to the muscle fibers which accordingly contract or relax the tissue. This system is indeed much more complex containing various

layers of signal tuning and more importantly forming a closed loop all together with the sensory neurons of cortex which receive feedback from the skin.

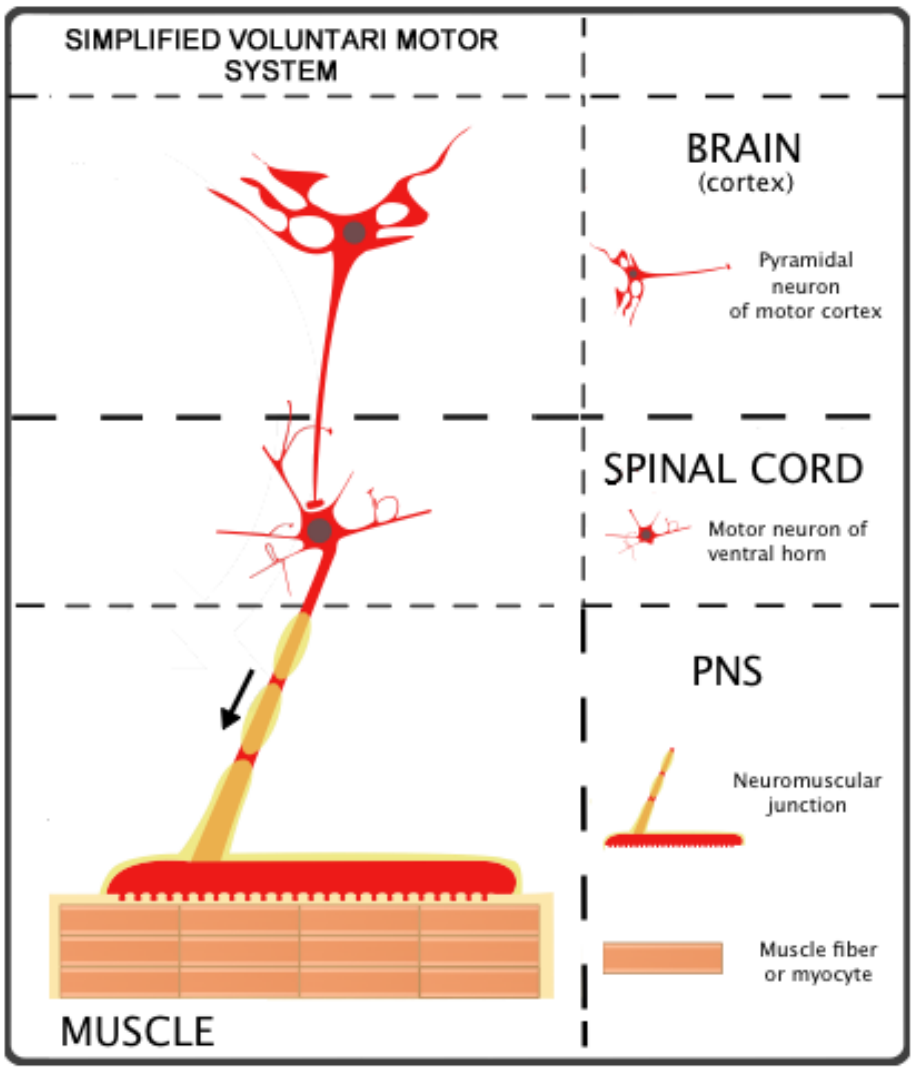

Figure 1.2: A simplified version of the human skeletal muscle control is shown starting at a neuron of the motor cortex issuing the signal and relying on the motor neuron of the spinal chord to deliver the signal to the muscle fibers through the neuro-muscular junction.

In Figure 1.2 we can see the simplified skeletal motor control system which is based on a single cortex neuron sending a signal, an *action potential* after all, to a muscle. While this system does conform the basic functional unit of voluntary muscle control it must be noted that a normal muscle, such as the biceps for example, is not controlled by a single cortex neuron but by a group of them. Each of this cortex neuron then sends a signal to a group of fibers in the given muscle, the bigger the muscle, the more cortex neurons become involved in the contraction and relaxation process. This entity introduced in Figure 1.2 shows what is called *motor unit* (MU), which is the part of the system starting at the

spinal cord and ending at the muscle fibers. This entity, the MU, is the basic unit that comes into action when a muscle contraction or relaxation is performed. As it is shown in Table 1.1 different muscles have a different number of motor units (which also varies from individual to individual) the number of which that come into action during muscle contraction, and the way they do it, having an impact on the magnitude of the force.

Table 1.1: This table shows the approximate number of motor units present in different muscles. It also states the approximate number of muscle fibers present and the innervation factor, the quotient of the last two, showing the average of how many fibers are attached to each MU.

Muscle	Number of MUs	Number of Fibers	Innervation Ratio
Medial gastrocnemius	579	1,120,000	1,934
Biceps	774	580,000	750
Tibialis anterior	445	250,200	562
Brachioradialis	315	129,000	410
First dorsal interosseous	119	40,500	340

Another characteristic of motor units which deserves some attention is the nature of the signal that eventually reaches the group of fibers and how it's characteristics affects the nature of the force produced. This signal is basically different for each motor unit and takes the form of a series of periodic pulses which are called *motor unit action potential trains*, (MUAPT from now on). While the amplitude of the signal is not relevant to the end effect it is their period which impacts the degree of the fiber contraction. Therefore, for a single muscle, a group of motor units exist each of those producing a signal compound of a train of pulses (more or less close together according to the force desired) to different groups of muscle fibers.

It is also worth noting that, even though the periodicity and number of signals which transmit orders from the brain cortex and eventually conform the MUAPTs, are linear and time-invariant to the force exerted by the subject (not accounting for muscle fatige, muscle growth, and other more subtle details), the proportionality constant is relative to each individual, since the specific strenght of muscle fibers can vary greatly among different subjects. In other words, the same number and frequency of MUAPTs might produce a greater or smaller forces depending on the muscle they innervate, since a big muscle will be conformed from bigger fibers, capable of exerting more force with the same stimulus.

1.2.3 EMG Measurements

The signals generated by each motor unit eventually reach the muscle tissue and excite the corresponding fibers. This signals then propagates longitudinally along those muscle fibers. Reached this point there are different ways to capture this signals. If a single motor

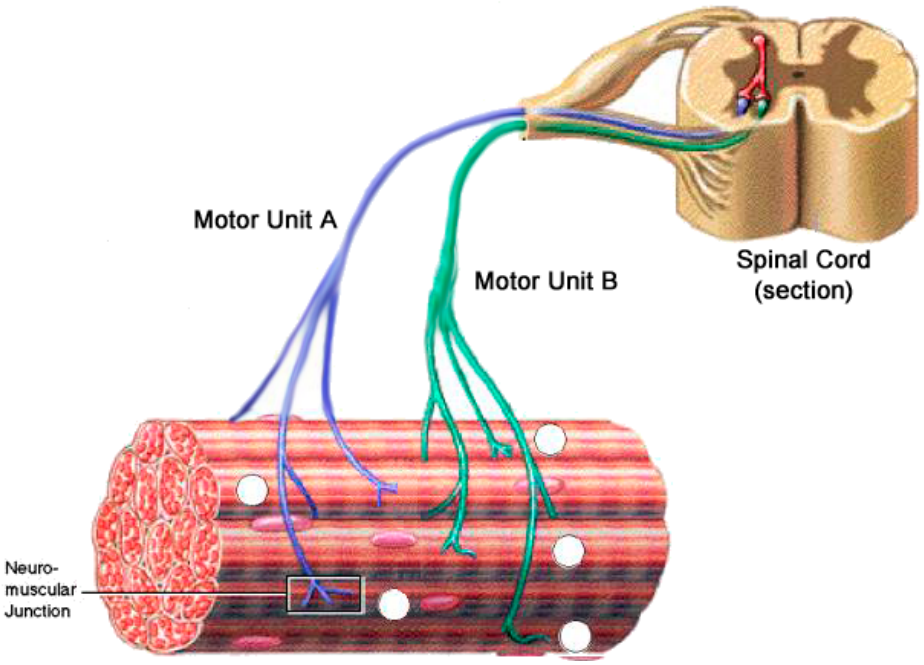

Figure 1.3: Ilustrates how multiple motor units excite different groups of fibers within a same muscle.

unit is to be measured, an electrode, usually in the form of a needle, needs to be inserted as close as possible to the group of fibers composing the specific motor unit. This type of EMG is usually referred to as *needle EMG* or *invasive EMG*. While this method of analysis allows for a much higher precision and motor unit isolation it is invasive and unnecessary if all the measurement is aiming is at evaluating the overall muscle activity.

Another type of EMG measurement exists in those cases, called *surface EMG* (sEMG from now on) which is based on the placement of a surface differential electrode over the skin covering a certain muscle. The obvious advantage of this method is it is non-invasive, however, it's nature prevents single motor unit activity from being isolated. Despite some research and even specific commercial software exists which attempts to isolate single motor units from sEMG measurement [14] [16] their methods always rely on probabilistic analysis requiring many electrodes to be placed during the reading and intensive post-processing.

In figure 1.4 we can see a simplified scheme of the MUAPTs addition which eventually is the signals that can be captured by an sEMG electrode.

Raw sEMG signals are usually within 20 and 500 Hz and they are used in plenty of cases where a detailed treatment of the signal is needed. For example, the methods previously mentioned which attempt at isolate single motor units (or at least get a magnitude of the number of them present) rely on the processing of this raw signal. However, for the aim of this thesis only a reading of the overall muscle activation is required, meaning that not all the subtle oscillations in the raw signal are necessary. In this cases special electrodes exists which not only gather the raw sEMG signal but also rectify and filter it giving a

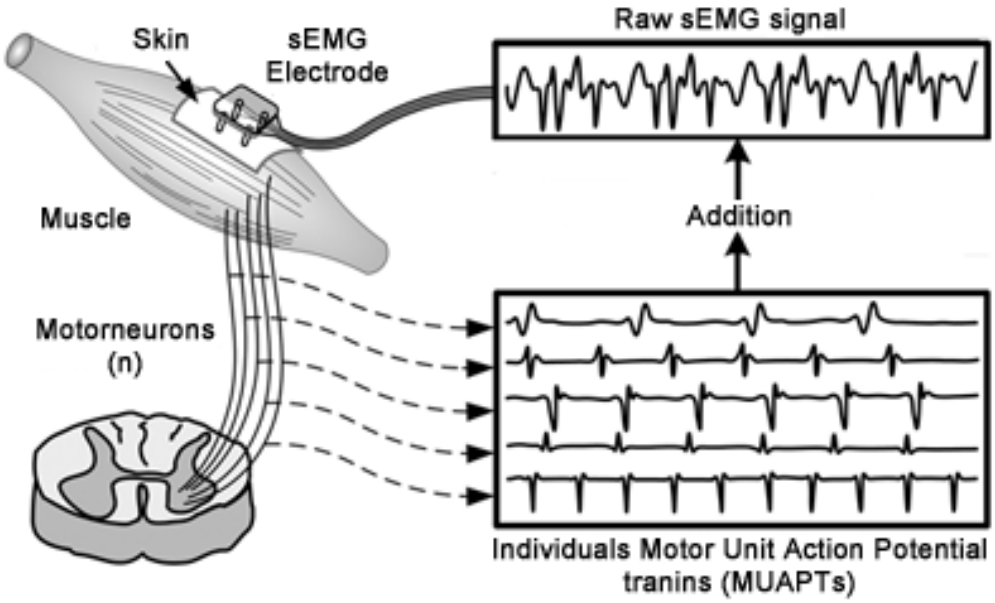

Figure 1.4: Illustrates how the signal measured by an sEMG electrode is the addition of each MUAPT which are exciting the muscle at a certain instant in time.

smooth, low-frequency, uni-polar output. These type of electrodes are the ones used in this work, their only drawback being the delay introduced by the low-pass filtering.

In spite of the variety of methods existent which post-process raw sEMG signals with similar purposes there is a pretty standard chain of transformations that, with more or less subtleties are applied to the signal. in Figure 1.5 we can see those most common stages which the signals passes through. The process is very similar to an AM (Amplitude Modulated) signal demodulation [5].

1.3 Overview

This project has been carried out at the Robotics and Mechatronics Center of the Deutsches Zentrum für Luft- und Raumfahrt (DLR) in Oberpfaffenhofen, Germany, during the last months of 2012 and the beginning of 2013. More specifically the project has been managed by both the Bionics Group and the Telepresence Group of this center. The first one deals with topics such as biomechanics and body-machine interfaces and is engaged in achieving advancements in the fields of rehabilitation and prosthetics. The second one deals mainly with teleoperation and is involved in projects relating On Orbit Servicing among many others.

This document intends to give a detailed description of the work carried out during this Master's Thesis. Therefore, this document should allow the reader to understand all

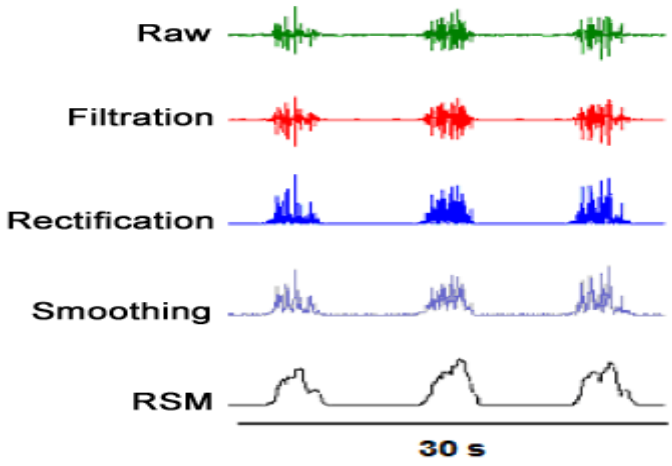

Figure 1.5: Illustrates the multiples stages a raw sEMG signal passes through when only it's envelope is of relevance. First the signals is filtered to reject noise and other components outside of the relevant EMG bandwidth, then the signal is rectified to obtain an unipolar version of it, then another phase of filtering is introduced due to reject new out-of-band frequencies introduced by the rectification and finally a curve fitting algorithm is applied (which still could be considered as a sort of filter), usually based on the Root Mean Average (RMS from now on)

the methods and conclusions reached and also serve as a written record and recompilation of all the results and tools used and implemented.

The text is divided in 5 sections. The first section contains the introduction, where the general theory for both teleoperation and sEMG is presented. The state of the art is also presented in this section through the reference of related works. Following, a detailed but brief description of the problem statement is presented in section 2, this section also contains a general description of the approach followed towards achieving the project's goal. Section 3 then goes on to present the whole technical setup where the experiments were performed. Even though a detailed description of all the elements conforming the setup is given and the main software written is also available in the appendices it is impossible to document a complex setup and group of experiments down to the last detail, for any further clarifications please contact the author. After this results are presented in section 4, numerous graphs have been included, however, the large amount and the very own nature of the data render unfeasible for it to be included in this thesis, again, for further details please feel free to contact the author or any of the supervisors involved.

1.4 Motivation

The ultimate goal in a telepresence system is to maximize transparency, that is, allowing the subject governing the master device, the operator, to feel as if they were present at a different location. In order to accomplish this many sorts of feedbacks such as vision or position can be used along with a unilateral or bilateral architecture. The latter one, force feedback, has been proven of crucial importance in any interaction with the remote

environment, to the point that those systems incorporating it can easily outperform those where it is not present at virtually any teleoperation task [6].

In these systems, the ability to reproduce the interaction forces at the operator's side effectively closes the loop, posing challenges concerning the balance between transparency and stability, and therefore placing constraints in the design of the coupling between master and slave, the *controller*. With transparency at mind, it is clear that the optimal mechanical linkage between the operator and the environment will be such that reproduces the interaction forces present at the slave's side back at the master's side in a faithful manner (or the other way around for that matter, notice that in a bilateral system the roles of master and slaves become relative). This behavior can be depicted as that of an infinitely light and infinitely stiff rod as can be seen in Figure 1.6. Nevertheless, multiple factors, like measurement errors or delays, render this ideal scenario impossible in a real setup. Accordingly the controller's response will have to be tuned in order to guarantee the system's stability and at the same time maximize transparency.

Figure 1.6: Depicts the optimal simplified behavior of a bilateral teleoperation system. Credits [23]

At this point some discussion concerning the ideal tuning of the controller arouses. While it seems that the highest value of stiffness[2] would be desirable, as it would allow the system to perform as if the controller was not present, (see the mechanical analogy in Figure 1.7) some applications, like for example those involving high delays, could benefit from a more adaptive approach.

While stiffness estimation has been used before in unilateral system as a sort of "tele-impedance" [2], this thesis takes on a different approach. While dynamic impedance tuning is certainly not as necessary in a setup equipped with force feedback as it is in one which is lacking it, this thesis intends to use it in order to create a more ergonomic and maneuverable system. More specifically, this is done by estimating the human arm stiffness and then tuning the controller accordingly, so that the controller will somehow mimic the dynamics of the human arm. This idea is illustrated with Figure 1.8. The way to achieve this will be to conceive and implement a system where the controller dynamically tunes it's stiffness to mimic that of the human arm's.

[2]In a dynamic system controllers usually have velocities and positions as inputs, this allows us to conceive them mechanically as spring-damper systems with a certain stiffness and viscosity

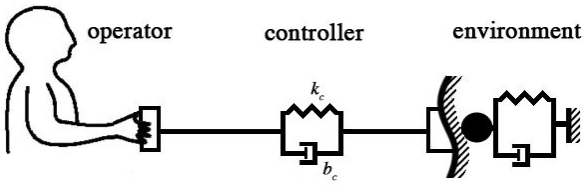

Figure 1.7: Depicts the mechanical equivalent to a common position based control scheme used in a bilateral teleoperation system. In the picture k_c, in units of $^N/_m$, and B_c, in units of $\frac{N \cdot s}{m}$, would determine the stiffness and the viscosity of the controller respectively. Credits [23]

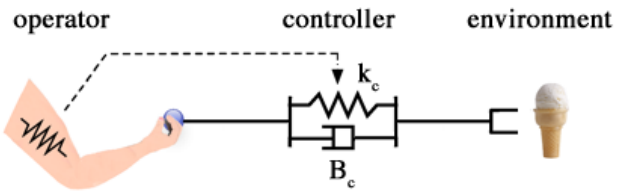

Figure 1.8: Illustrates the concept of an sEMG based tuning of a controller's stiffness.

Whether or not the technic described in this thesis will provide any real improvement on a teleoperation system remain unclear. The corresponding evaluations would involve a large number of considerations and they would depend both on the subject, and even more importantly, on the task been considered. It is the authors opinion however, that given the numerous fields in which teleoperation is applied today, some combinations of tasks and environments will for sure be able to benefit from such approach.

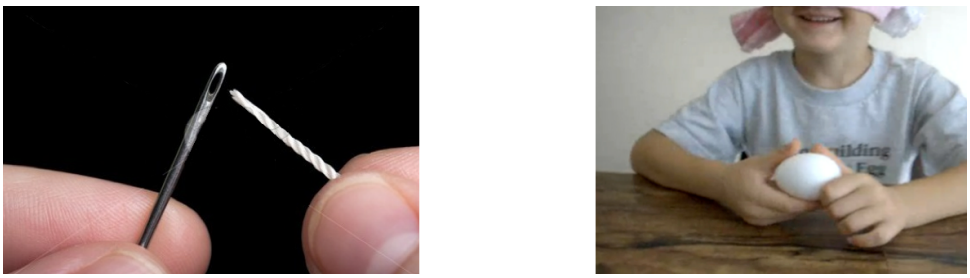

Figure 1.9: To the left a task which, due to precision, requires a high limb stiffness. To the right, a task which, due to uncertainty, requires low lib stiffness.

For the sake of clarity Figure 1.9 two examples where very different impedances are used by a human subject in order to perform a certain task. In one we can see a delicate and unknown environment where the operator has to tune his limbs with a low stiffness in order to avoid harmful dynamic interactions while in the other we have a task requiring the

operator to use a very high stiffness since the task at hand is not so mechanically fragile but much more precision demanding.

2

Related Work

To accomplish the established goal analysis on the basic teleoperation schemes first needs to be conducted. First of all the generic architecture of a bilateral system will be introduced and then a parametrization of teleoperation bilateral systems will have to be reached. In order to be able to tune the system to behave in a different and desired manner specific parameter or parameters will need to be identified and finally the bound of such parameters stablished.

2.1 Teleoperation

As already introduced teleoperation, or tele-manipulation in our case, is the act of manipulating an environment at a distance. Since physical environment are characterized by masses, damping, and stiffness, tele-manipulation systems can easily be described, given a set of initial conditions, as Linear Time-Invariant. Along with the Laplace Transform (LT from now on) defined by Equation 2.1 and the Inverse Laplace Transform defined by Equation (ILT from now on) 2.2 this systems can be represented compactly and simply from a mathematic standpoint like any other control system.

$$F(s) = \mathcal{L}\left\{f(t)\right\}(s) = \int_0^\infty e^{-st} f(t)\, dt \ , \ \forall\, s \in \mathbb{C} \tag{2.1}$$

$$f(t) = \mathcal{L}^{-1}\{F(s)\} = \frac{1}{2\pi i}\lim_{T\to\infty}\int_{\gamma-iT}^{\gamma+iT} e^{st}F(s)\,ds\ ,\ |\ \gamma \in \mathbb{R}\ \ \&\ \ \exists\, F(\gamma+i\beta)\,\forall\,\beta \qquad (2.2)$$

Using these tools we can transform an initially complex differential system to a simple polynomial and then, once performed the required analysis and modifications undo the transformation to recover the initial physical quantities. To be able to grasp the usefulness of this procedure an example concerning a simple automobile suspension system follows[21].

Although much more complex mechanisms exists many wheel suspension systems are based on a damper and a spring. A damper is such and element which behaves similarly as friction, applying a force proportional to the relative speed at it's terminals and always in a direction agains the direction of movement. A spring, an object commonly known, is an element which behaves as a stiffness, applying a force proportional to the relative displacement of it's terminals and in a direction always towards it's position of equilibrium. Both damper and spring have a point of equilibrium, which is a point at which no force is exerted by the element, however, these two equilibrium points differ in their nature. While a spring has a position as a point of equilibrium, a damper has a velocity. More importantly, the fact that a damper always exerts a force against the instant velocity of displacement, allows such element to dissipating energy. While other physical elements like springs and masses store potential energy and release kinetic energy, a damper always introduces an energy loss. This becomes obvious by analyzing the definition of energy as the integral of the force times de displacement as in Equation 2.3.

$$E = \int_a^b F(x)\,dx \qquad (2.3)$$

A simple suspension system as described can be depicted like it is shown in Figure 2.1. As illustrated such system is composed by the mass of the vehicle which is obviously under the effect of the gravitational field, a spring, and a damper.

The classic temporal domain equation describing the behavior of the system is an ordinary differential equation. If y_0 is assumed to be the elevation of the chassis while at rest (nothing more than initial condition), by balancing the forces at both ends of the system Equation 2.4 can be reached.

$$M \cdot \frac{d^2y(t)}{dt} + b \cdot \frac{dy(t)}{dt} + k \cdot y(t) = k \cdot x(t) + b \cdot \frac{dx(t)}{dt} \qquad (2.4)$$

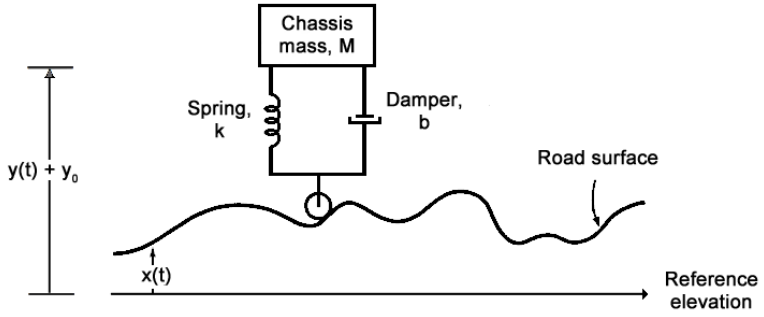

Figure 2.1: Illustrates a simplified version of a vehicle's suspension system with a mass of M units of mass, k units of Newtons per meter and b of Newtons per meters per second.

Of course an expression could be found from Equation 2.4 establishing the relationship between the input function $x(t)$ and the output function $y(t)$. However this would involve solving a differential equation, while, if this same equation is transformed using the Laplace definition introduced in Equation 2.1, the same system can be characterized using the Laplace transformation with Equation 2.5.

$$M \cdot \frac{F_y(s)}{s^2} + b \cdot \frac{F_y(s)}{s} + k \cdot F_y(s) = k \cdot F_x(s) + b \cdot \frac{F_x(s)}{s} \tag{2.5}$$

Reached this point it is trivial to find the expression governing the system which is shown in 2.6.

$$H(s) = \frac{F_y(s)}{F_x(x)} = \frac{k + b \cdot s}{M \cdot s^2 + b \cdot s + k} = \frac{{w_n}^2 + 2\xi w_n s}{s^2 + 2\xi w_n s + {w_n}^2} \tag{2.6}$$

Where, to ease up the following analysis most textbooks attach to the convention that $w_n = \sqrt{k/M}$ and $2\xi w_n = b/M$. By replacing the variable s by $jw + \alpha$ it can be seen how the system would react to different road profiles, to be precise, we can see how the system would react to any road profile which could be expressed by the multiplication of an sinusoidal oscillation and an exponential pattern. In Figure 2.2 it can be seen the results of such method when s is made to be purely imaginary, so that the road presents a purely sinusoidal pattern.

Figure 2.2 depicts the response of the system, in other words, the function $H(s)$ for $s = jw$. This kind of plot is usually called a bode plot in honor of Hendrik Wade Bode who ideated and made public a certain number of approximation methods which greatly simplified the representation of polynomial fractions such as $H(s)$. In such figure it can be seen how the response of the system can be modulated according to the previously defined parameters. As it can be seen in the plot the system presents a peak response at w_n which

is inversely proportional to ξ. At the same time, the steepness of the response is also tied to this last parameter, this time directly proportional to it. Therefore a compromise exists between the magnitude of the peak response and the steepness of the same. While a high ξ would involve a low peak at w_n, meaning that the behavior of the system at that frequency would not be so dramatic, it would also involve a less steep response, meaning that higher frequencies above w_n would suffer a smaller attenuation, therefore making the suspension more stiff, in other words, it would transmit fast changing values of the road surface to the car's chassis, an effect that, in general, is not desired. On the other hand a low value of ξ would provoke a higher peak response at w_n but at the same time a steeper response for higher frequencies, which would be better filtered out.

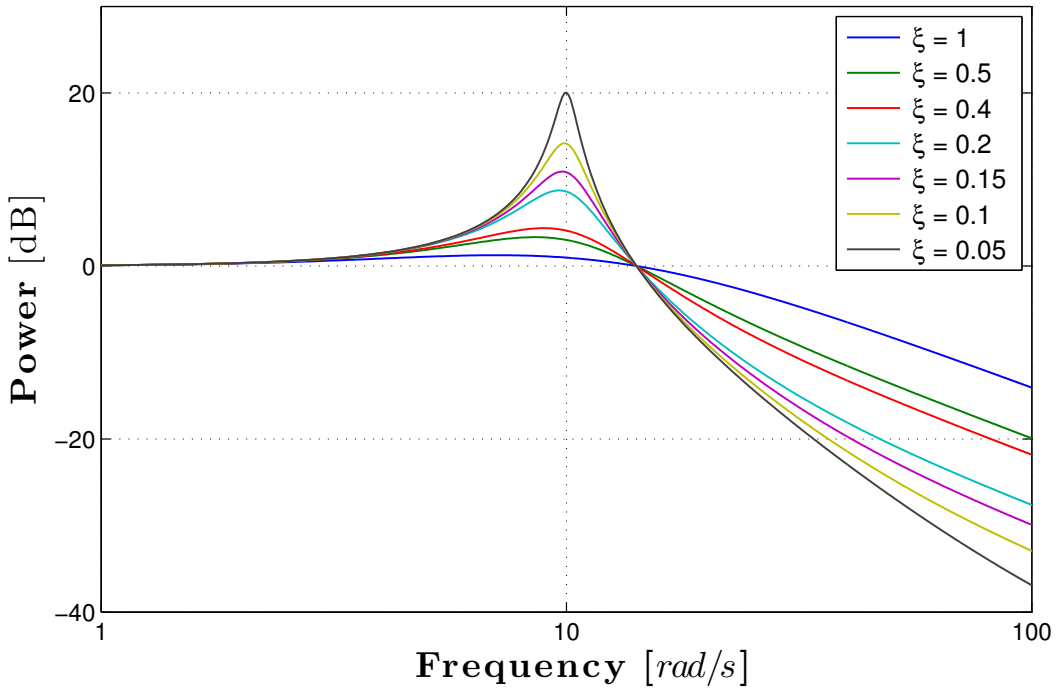

Figure 2.2: Here the frequential response of the system is determined for varying values of the parameter ξ. As it can be seen a compromise exists between the steepness of the frequency response and the magnitude of it's peak.

There is however another method to analyze systems which will be extremely useful when dealing with more involve models. The *zero-pole diagram* consists of a complex plane where the zeros (roots of the numerator in $H(s)$) and poles, (roots of the denominator in $H(s)$) are indicated and their trajectory according to a certain parameter is traced. Observating the different positions along their trajectory many useful conclusions can be extracted like stability or transparency, this method, specially when applied to certain parameters, receives the lame *root locus*. If this latter method is used to analyze the same suspension system and we take ξ as the varying parameter the graph depicted in Figure 2.3.

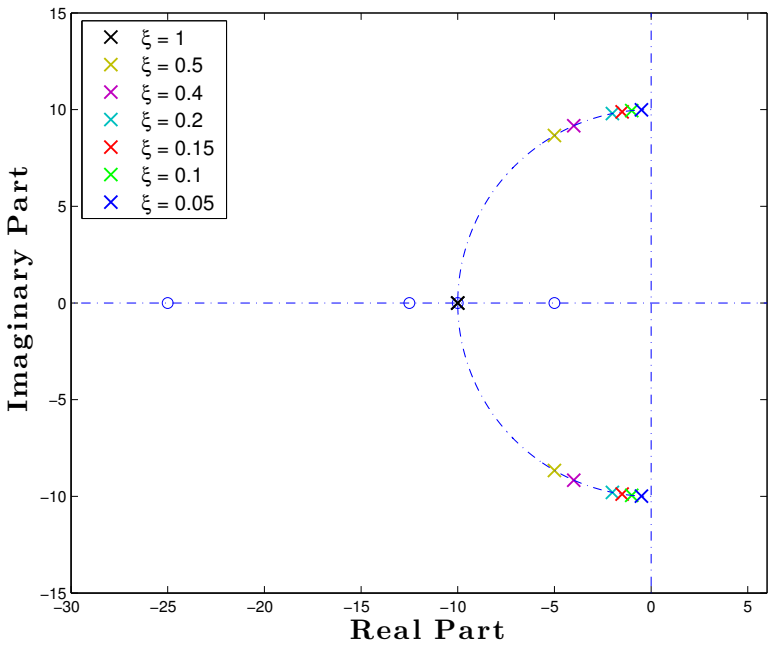

Figure 2.3: Here the characteristics of the system are represented with a complex plane where the roots of $H(s)$ have been plotted as ξ takes on different values.

It can be shown that a second order system such as the one discussed present superior performance when the denominator's roots meet at $-w_n$, this happens precisely for $\xi = 1$. If this parameter was increased beyond unity one of the poles of $H(s)$ would move closer to the plane's center, making the system slower the closer it got to it.

When this particular case is met, ξ equals unity, the system is said to be *critically damped*. This name stems from the fact that it is at this point where the system, given an "step-like" discontinuity, would react in the quickest way possible within the restriction of not oscillating afterwards. In the present example case such a step function would mean the road presented a profile where it's height increased by a finite amount abruptly.

As a final note on this example it is interesting to see how easily such systems, once transformed to the Laplace domain, can be simply represented. If we take Equation 2.6 it can easily be seen that,

$$F_y(s) = F_x(s) \cdot \frac{{w_n}^2 + 2\xi w_n s}{s^2 + 2\xi w_n s + {w_n}^2} \tag{2.7}$$

Which can then be graphically represented in any of the schemes shown in Figures 2.4 and 2.5.

These representation, even though given the current system might seem trivial, are of great importance when dealing with more intrinqued systems. It is important not to

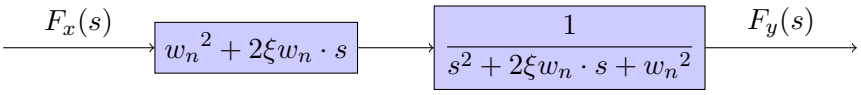

Figure 2.4: Depicts an equivalent representation by placing numerator and denominator in series forming the same $H(s)$ obtained previously

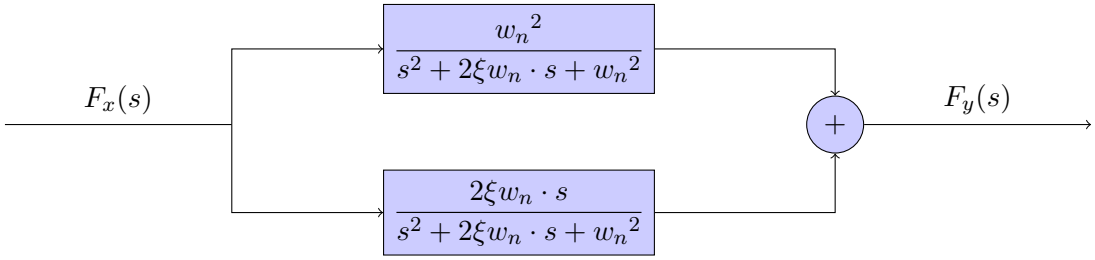

Figure 2.5: Depicts an equivalent representation by splitting the original transfer function $H(s)$ in two equations which, when added, are equivalent to the latter one.

forget that all these representations are so easy to come because of the polynomial nature of the transfer function $H(s)$, had the original differential equation been the basis of the analysis all these representations might have not been trivial at all.

2.2 Bilateral Control

Now that the main tools have been introduced it is now time to focus further on the topic at hand. Bilateral control systems are those in which output and inputs exists at least at one "end" of the system. As it will be seen in the following sections in a tele-manipulation system this might involve considerable complications as in any imaginable tele manipulation scenario these outputs and inputs will end up forming a closed loop with all the stability concerns this might bring along. Therefore one could simply define a bilateral system as shown in Figure 2.6.

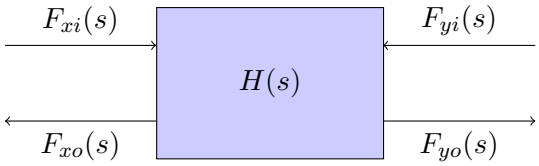

Figure 2.6: Depicts the maximum simplification of a bilateral system, understanding such as one where the "information" flows in both directions.

2.2.1 System Architectures

In this section the main system architectures existent for a bilateral telemanipulation control system will be introduced. As it will be seen the only difference among different architectures are the sampled variables.

Position-Position Architecture

As the architecture's name states this rely's on the positions of both master and slave as sampled and replicated variables. The scheme for such architecture can be seen in Figure 2.7.

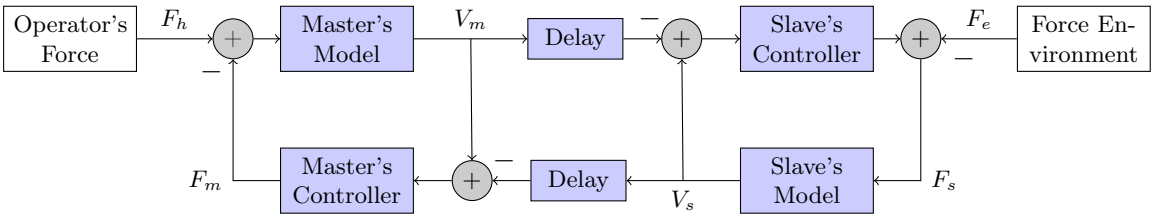

Figure 2.7: Illustrates a bilateral telemanipulation architecture based on the sampling and replication of positions on both sides of the system. The signals labeled with "F" stand for forces and the ones labeled with a "V" stand for velocities.

What Figure 2.7 depicts is the generic concept of a bilateral position-position telemanipulation system. Note that, even though the variables transmitted sent and received are velocities, which usually would be the sampled variables, the figure has forces as inputs.

As for the different models and controllers, the master and slave's blocks are usually modeled by a mass and a damping coefficient. A solution which is pretty reasonable if we think about it as those physical qualities are the ones that are, in principle, unavoidable when building an automaton. As for the controllers, such entities can be as complex as one might what to conceive, although this thesis's scope is limited to those belonging to differential first order, by that meaning those incorporating a coefficient which is multiplied by the original variable, the position, and another value, which is to be multiplied by the derivative of this magnitude, the velocity. The term, *first order* becomes clear once the transformation to the Laplace space of such entities is viewed, as the resulting polynomials are first order polynomials of s. Figure 2.8 depicts a system based on the models just stated.

A subcategory of position-position bilateral systems are those called *symmetric*, meaning that both controllers and devices (master and slave) are identical. However this might not be always the case. In some scenarios it is advantageous being able to tune different controllers for master and slave. On the other hand the given architecture can present some weakness if the master and slave are not physically identical as then some spatial conversion according to the forward and backward kinematics of them both would

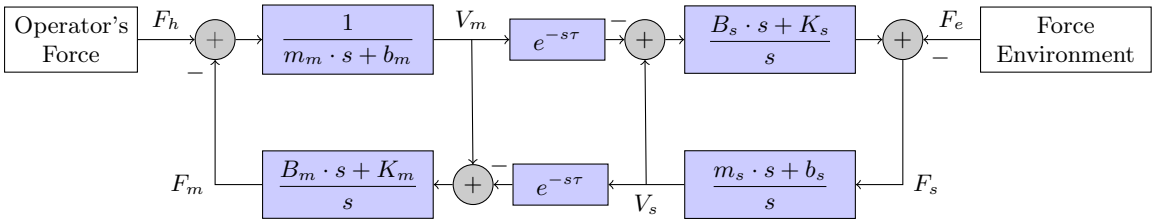

Figure 2.8: Illustrates a bilateral telemanipulation architecture based on the sampling and replication of positions on both sides of the system. The signals labeled with "F" stand for forces and the ones labeled with a "V" stand for velocities. In this case the generic blocks have been substituted by the commonly used models of a master and slave automaton and first order controllers.

need to take place before the actual forces to be applied can be computed. Figure 2.9 depicts an example of a symmetric position-position bilateral system, as it can be seen all the constants in both automatons and controllers have been replicated yielding a much simpler system.

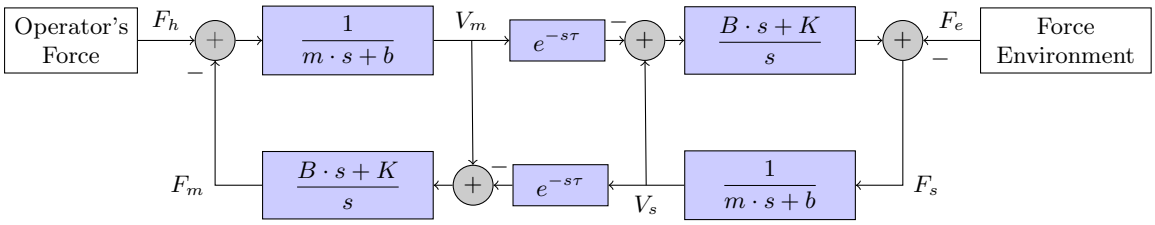

Figure 2.9: Illustrates a bilateral telemanipulation architecture based on the sampling and replication of positions on both sides of the system and symmetric models and controllers at both sides of the system.

Position-Force Architecture

Another widely used bilateral control architecture is one called the *Position-Force Architecture*. As the name states this architecture is based on the sampling of a position at one side of the system and the replication of a force at the other extreme of the system. As it will be seen in following sections, under certain conditions this architecture shares a lot of the same characteristics with the Position-Position Architecture. Figure 2.10 depicts the basic structure of such systems.

As it can be seen from the diagram in this case the system relays on a single controller, this, while adding simplicity, can also limit the capabilities of the system in some cases.

Just like in the previous cases, if the different blocks are substituted by the corresponding Laplace models and the assumption of symmetry the diagram depicted in Figure 2.11 can be reached.

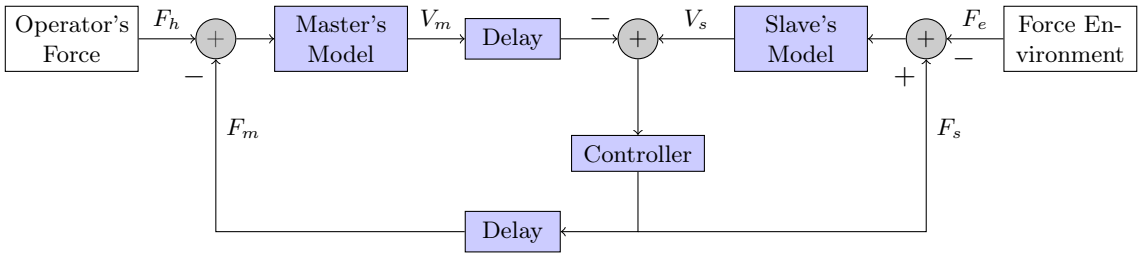

Figure 2.10: Illustrates a bilateral telemanipulation architecture based on the sampling and replication of a position in one direction of the system and the sampling and replication of a force in the other direction of the system. The signals labeled with "F" stand for forces and the ones labeled with a "V" stand for velocities.

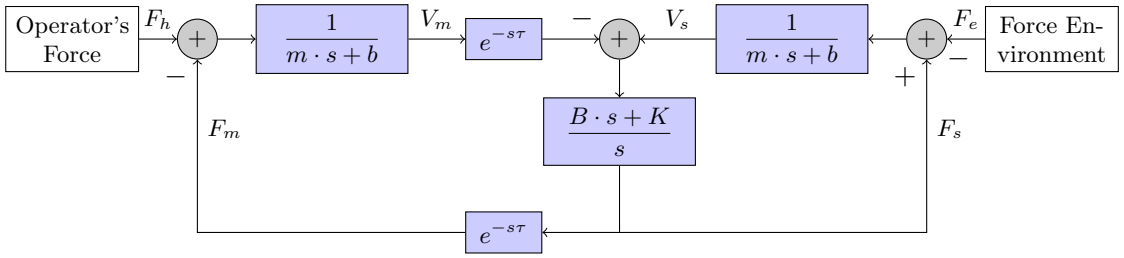

Figure 2.11: Illustrates a bilateral telemanipulation architecture based on the Position-Force Architecture where the different entities have been modeled according to their Laplace transformation

2.2.2 Performance and Stability

In this section the different architectures introduced will be analysed and the controller's theoretical optimum parameters will be presented. It's important to note that first all system delay's will be suppressed, leading to, as it will be seen, to much simpler systems. After this section a whole other section will be devoted to analyzing the effects of such delays and the huge impact those can have on the overall performance and stability will become apparent.

By suppressing the forward and backward delays from the presented architectures the previous schemes become second order systems depicted by Figure 2.12 for the Position-Position Symmetric Architecture and Figure 2.13.

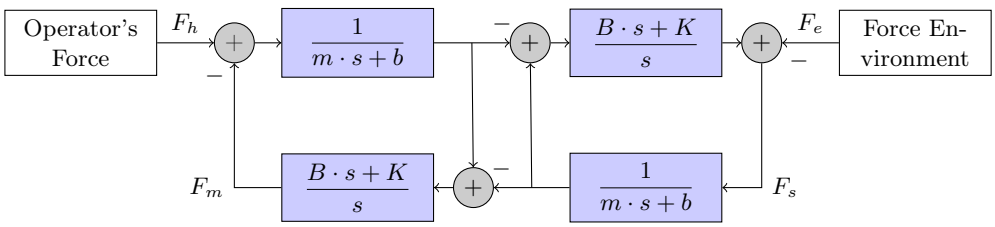

Figure 2.12: Illustrates a bilateral telemanipulation architecture based on the sampling and replication of positions on both sides of the system and symmetric models and controllers at both sides of the system where there is supposed to be no delay in the transmission and reception of the variables.

Thanks to the suppression of delays it can easily be seen that both architectures (Position-Position and Position-Force) can be simplified to the same expression. Assuming

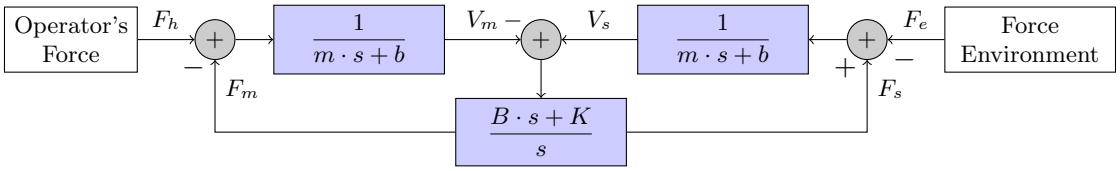

Figure 2.13: IIllustrates a bilateral telemanipulation architecture based on the sampling and replication in one direction of the system and the sampling and replication of a force in the other direction where there is supposed to be no delay in the transmission and reception of the variables.

the environment is not exerting any force the transfer function for $H(s) = {}^{F_h}/_{F_m}$ can be seen in Figure 2.14.

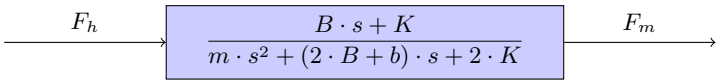

Figure 2.14: Represents the overall system scheme in open-loop form for a delay-free Position-Position or Position-Force architecture. Capital letters stand for the controllers parameters and lower-case letters for the automatons parameters (masses and damping factors).

Allthough this is the transfer function of the system, in order to carry out stability analysis it is usually preferable to reach a closed loop equivalent, which, in fact, is the original simplification one would come across before if the simplification of schemes 2.13 or 2.12 was to be done step by step. Figure 2.15 represents the system in such form.

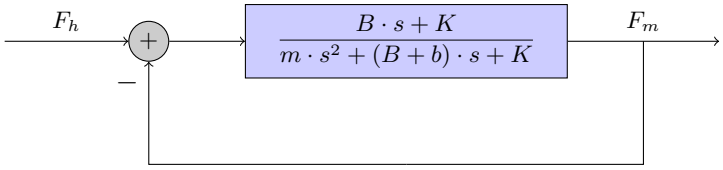

Figure 2.15: Represents the overall system scheme in closed-loop form for a delay-free Position-Position or Position-Force architecture.

A system is said to be stable when no bounded input exists capable of producing an unbounded output. Many methods exist to determine stability from the open-loop expression of a given system. One of these methods is called the *Routh-Hurwitz* method [22]. Even though the insights of this and many other methods are not trivial they ultimately aim at determining if the final closed-loop expression will have any roots within the right half part of the complex plain, therefore yielding an increasing exponential in their impulse response or not. In table 2.1 we can see employment of such method.

This method guarantees the system will be stable if there is an even change of signs in the left-most column right of the vertical line. It can easily be understood that the first item on the column, the mass parameter of the automaton will never be negative, as it is a mass. Also, the last element of the column $2 \cdot K$ should be positive, as it makes little sense

Table 2.1: Here the development of the method described by Edward John Routh and Adolf Hurwitz.

s^2	m	$2 \cdot K$
s	$(2 \cdot B + b)$	0
	$2 \cdot K$	

to invert the direction of the motion. In order to obtain an stable system then, the second term of the column must be positive, therefore an stability condition can be stablished which is stated in Equation 2.8.

$$B \geq {}^{-b}/_{2} \tag{2.8}$$

Even though the conditions for stability are now known the previous analysis does not give or suggest any information on specific values in order to obtain the best possible performance. In order to obtain such information a different kind of tool is available. Root-Locus is a method which allows to graphically see the evolution of the system's response as different parameters (in this case the controller's) take on different values. Applying the Root-Locus method to a system with two variables is, however, a little bit involved, such method involves finding two different equations and performing Root-Locus on both for latter "merging".

From figure 2.14 we can conclude that the closed-loop equation governing the system is the one stated in Equation 2.9. Now, because of the negative feedback it can be stablished that the roots of such system (which basically determine the whole response of such) will be found where the condition in Equation 2.10 is met. However, as previously stated, the system is still under-constrained as the final location of the poles still depends on two variables B and K. However, if together with $H(s)$ and the condition established in Equation 2.10 expressions are isolated for both B and K expressions can be found for system's with equivalent poles where those variables B and K would be the single variables under analysis. With these equations the Root-Locus method can be exploited and then, with a little bit of insight, be able to grasp how the whole system's Root-Locus would evolve. Such development can be seen in Equations 2.11

$$H(s) = \frac{B \cdot s + K}{m \cdot s^2 + (B + b) \cdot s + K} \tag{2.9}$$

$$T(s) = \frac{H(s)}{1 + H(s)} \tag{2.10a}$$

$$poles \rightarrow 1 + H(s) = 0 \tag{2.10b}$$

$$H(s) = -1 \tag{2.10c}$$

$$B = -\frac{m \cdot s^2 + b \cdot s + 2 \cdot K}{2 \cdot s} \quad \rightarrow \quad G_B(s) = \frac{2 \cdot s}{m \cdot s^2 + b \cdot s + 2 \cdot K} \tag{2.11a}$$

$$K = -\frac{m \cdot s^2 + 2 \cdot (B + b) \cdot s}{2} \quad \rightarrow \quad G_K(s) = \frac{2}{m \cdot s^2 + 2 \cdot (B + b) \cdot s} \tag{2.11b}$$

Once this expressions have been reached the Root-Locus can be performed for both of them. In Figure 2.16 the results from $G_K(s)$ can be seen. As it is the "system's" (they are not really the system's equations anymore, they just behave as such when the loop is closed) poles converge from the plane's origin and minus infinity to a common purely real location and then evolve vertically in opposite directions. It is of common sense that the system should have a stiffness as high as possible in order to approximate as closely as possibly the ideal tele manipulation scheme presented in Figure 1.6. However, it can be seen from the plot that such high values would also provoke high resonation frequency.

If then a certain value of K is stablished and the same procedure is repeated with the expression $G_B(s)$ the final position of the system's poles can be acquired. As it can be seen in Figure 2.17 incrementing the value of B produces the poles to follow a circular trajectory converging in the real axis and taking on greater negative values. This can intuitively be seen as a very positive effect, as moving the poles closer to the real axis will produce a lower (or maybe even non-existent) peak at the resonation frequency and also it will mean that the system will become faster, since the more negative the poles of the system are, the faster the impulse response will become.

After seeing the system's evolution according to the controller's parameters a valuable result for a delay-free system can be obtained. As it has been proved, given a certain value of K there is a unique value of B which renders both poles real and equal, (making the system "critically damped" as introduced previously) eliminating the peak at the resonation frequency and making the system as "fast" as possible. It is of value to find the analytical expression giving the value of B which produces this effect given a certain predetermined value for K. To obtain such expression the value of B which produces

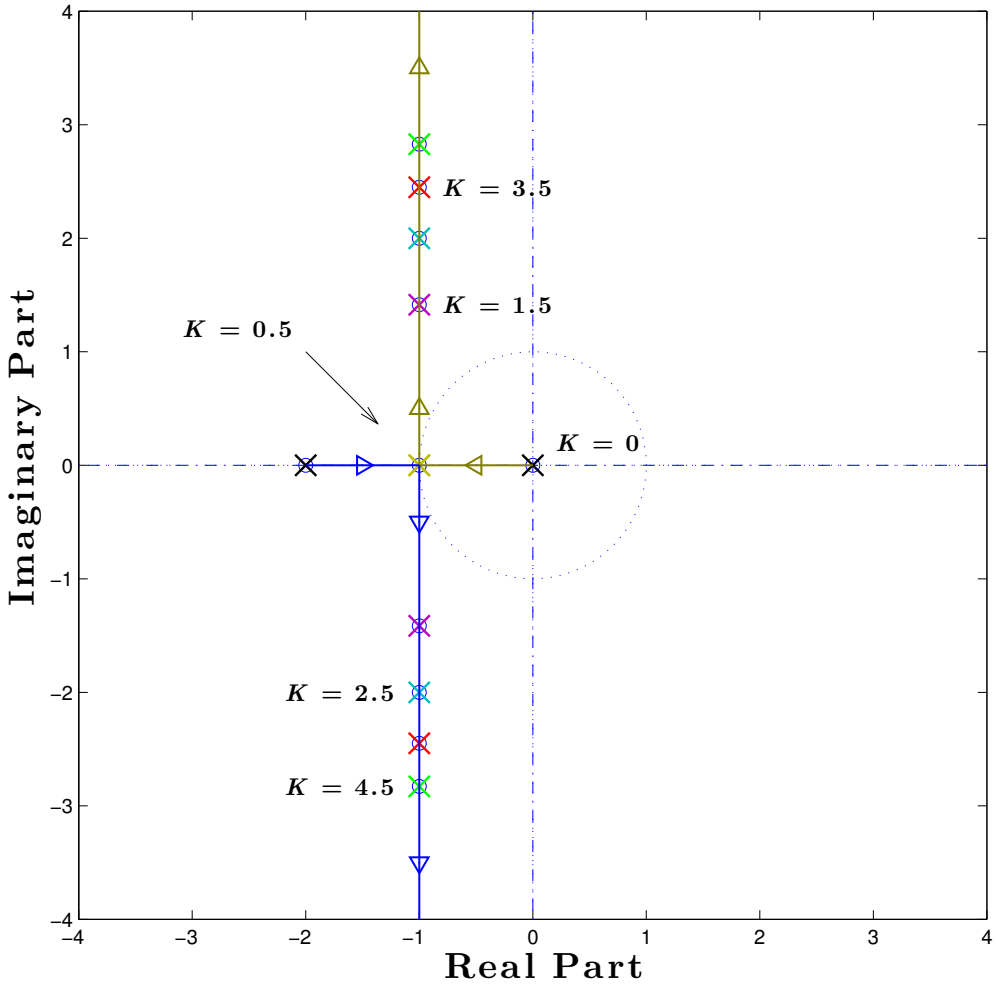

Figure 2.16: This image depicts the trajectory the system's poles follow according to the parameter K. To generate this graph values have been stablished as $m = 1$ as the mass of the automaton, $b = 1$ as the damping coefficient of the same. As for the parameter B a value of zero has been used.

the system's poles to be purely real is isolated, the procedure can be seen in Equation 2.12.

$$G_B(s) = \frac{2 \cdot s}{m \cdot s^2 + b \cdot s + 2 \cdot K} \tag{2.12a}$$

$$T(s) = \frac{G_B(s)}{1 + G_B(s)} \quad \rightarrow \quad \text{poles} = \frac{-(2B + b) \pm \sqrt{(2B + b)^2 - 8 \cdot mK}}{2 \cdot m} \tag{2.12b}$$

$$\text{poles} \in \Re \quad \rightarrow \quad \sqrt{(2B + b)^2 - 8 \cdot mK} \quad \geq \quad 0 \quad \rightarrow \quad B \geq \frac{\sqrt{8mK} - b}{2} \tag{2.12c}$$

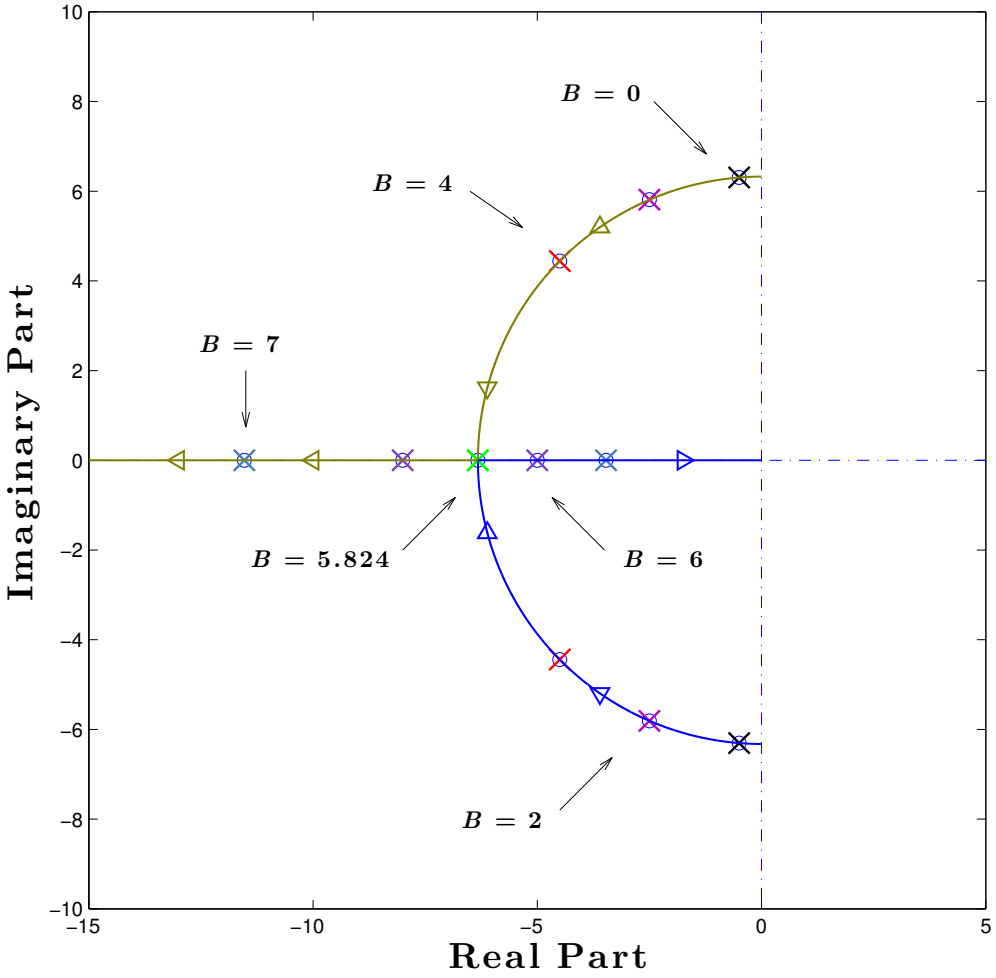

Figure 2.17: This image depicts the trajectory the system's poles follow according to the parameter B. To generate this graph values have been stablished as $m = 1$ as the mass of the automaton, $b = 1$ as the damping coefficient of the same. As for the parameter it has been assumed that $K = 20$.

As it can easily be seen the optimum value of B is nothing but a subgroup of the stability region defined by 2.8.

Besides the analysis just conducted which was concerned mainly in the position of the system's poles in the complex plane it is also of great importance to check the validity of the results in a more direct way. A good benchmark of such is the response of the system to a step input. If the hypothesis and reasonings presented until now are correct it should be found that the presented values are the ones which produce a critically damped response,

in other words, a resonse that neither oscillates or crosses the final value at any other point but infinity and also which converges to such in the quickest way possible.

Although many methods exist for obtaining such response a particularly elegant one would be to multiply the transfer function of the system by the transfer function of a unity step input and then transform the result to the temporal domain. This procedure is presented in Equation 2.13.

$$O(s) = T(s) \cdot U(s) = \frac{1}{s} \cdot \frac{B \cdot s + K}{(s - pole_1) \cdot (s - pole_2)} \tag{2.13a}$$

$$= \frac{1}{s} \cdot \frac{B \cdot s + K}{(s + \frac{(2B+b) - \sqrt{(2B+b)^2 - (8mK)}}{2m}) \cdot (s + \frac{(2B+b) - \sqrt{(2B+b)^2 - (8mK)}}{2m})} \tag{2.13b}$$

$$O(t) = \mathcal{L}^{-1}\{O(s)\} = \mathcal{L}^{-1}\left\{\frac{A(s)}{s} + \frac{B(s)}{(s - pole_1)} + \frac{C(s)}{(s - pole_2)}\right\} \tag{2.13c}$$

$$\rightarrow A(s) = \frac{B \cdot s + K}{s \cdot (s - pole_1) \cdot (s - pole_2)} \tag{2.13d}$$

$$\rightarrow B(s) = \frac{B \cdot s + K}{s \cdot (s - pole_2)} \tag{2.13e}$$

$$\rightarrow C(s) = \frac{B \cdot s + K}{s \cdot (s - pole_1)} \tag{2.13f}$$

By analyzing the equation reached in 2.14 that has been used all along it can be seen, however, that our system is not a normal second order system, understanding such as one with unity gain at frequency zero and a single constant as numerator. The system at hand however has two particular aspects that need to be accounted for. First we can easily see that the gain at zero frequency is not unity, but $1/2$, this can be easily understood by keeping in mind that this is a bilateral system and the transfer function presented is merely F_h/F_m and therefore only taking into account the force applied at one side of the system. Secondly, and more importantly we can se how the system does not have a single constant for a numerator, but a first order polynomial of s, $Bs + K$, this, regardless of it's presence's reason will disrupt (more or less according to the controllers parameters) the standard response of a second order system. It can be seen that the system transfer function can be split as shown in 2.14.

$$T(s) = \frac{s}{{}^{m}/_{B} \cdot s^2 + (2 + {}^{b}/_{B}) \cdot s + {}^{2K}/_{B}} + \frac{{}^{K}/_{B}}{{}^{m}/_{B} \cdot s^2 + (2 + {}^{b}/_{B}) \cdot s + {}^{2K}/_{B}} \tag{2.14}$$

By taking into account one of the Laplace Transform properties which states $\mathcal{L}\left\{{}^{df(t)}/_{dt}\right\}(s) = s \cdot F(s) - f(0)$ it can be seen that the first term is nothing more but the derivative of the second order system scaled by a factor of ${}^{K}/_{B}$. However, as it is shown in Equation 2.12 the optimum value of B is proportional to the square root of $8mK$, meaning that, for large values of K, which in fact are the desirable ones, it can be stated that $B << K$ and therefore ${}^{K}/_{B} >> 1$ rendering the mentioned perturbation irrelevant in the limit where K tends to infinity.

In Figure 2.18 the system's response to a step input can be seen. To generate the graph the previously deduced optimum values of B according to K have been used (as stated by Equation 2.12) except for the last of the legend's plot where a different sub-optimum value for B has been used. As expected an overshot followed by a converging oscillation can be seen in such case, this effect is not present when the value of B is chosen according to 2.12 so that the system is critically damped.

As it can be clearly seen from the graph the system can be made increasingly fast by increasing the value of K making the response more ideal with every increase. However, this will not be the case in a real system where there is always some delay, nor could it be achieved in an hypothetically delay-free system because of measurement errors and noise.

2.2.3 The Effect of Delay

Now that a simplified model has been studied it is time to analyze to what extent this model is affected when a delay is present in the communication path between the master and slave device, which, in practice, will always be the case.

In Figures 2.19 and 2.20 the scheme for a bilateral Position-Position Symmetric architecture and a Position-Force are presented once again.

The first difficulty when analyzing such schemes is that in contrast to what happens when the transmission delays are suppressed, the Position-Position Symmetric scheme and the Position-Force scheme do not present the same behavior. This can be certified by simplifying the previous schemes to their basic expression, while in the Position-Force system both delays end up in the same loop, allowing them to be combined into a single delay twice as long this does not happen in the Position-Position Symmetric scheme.

Such developments can be seen in Figure 2.21 for the Symmetric Position-Position scheme and in Figure 2.22 for the Position-Force scheme. While not impossible, it becomes obvious that the analysis of stability and performance of this system is much more involved

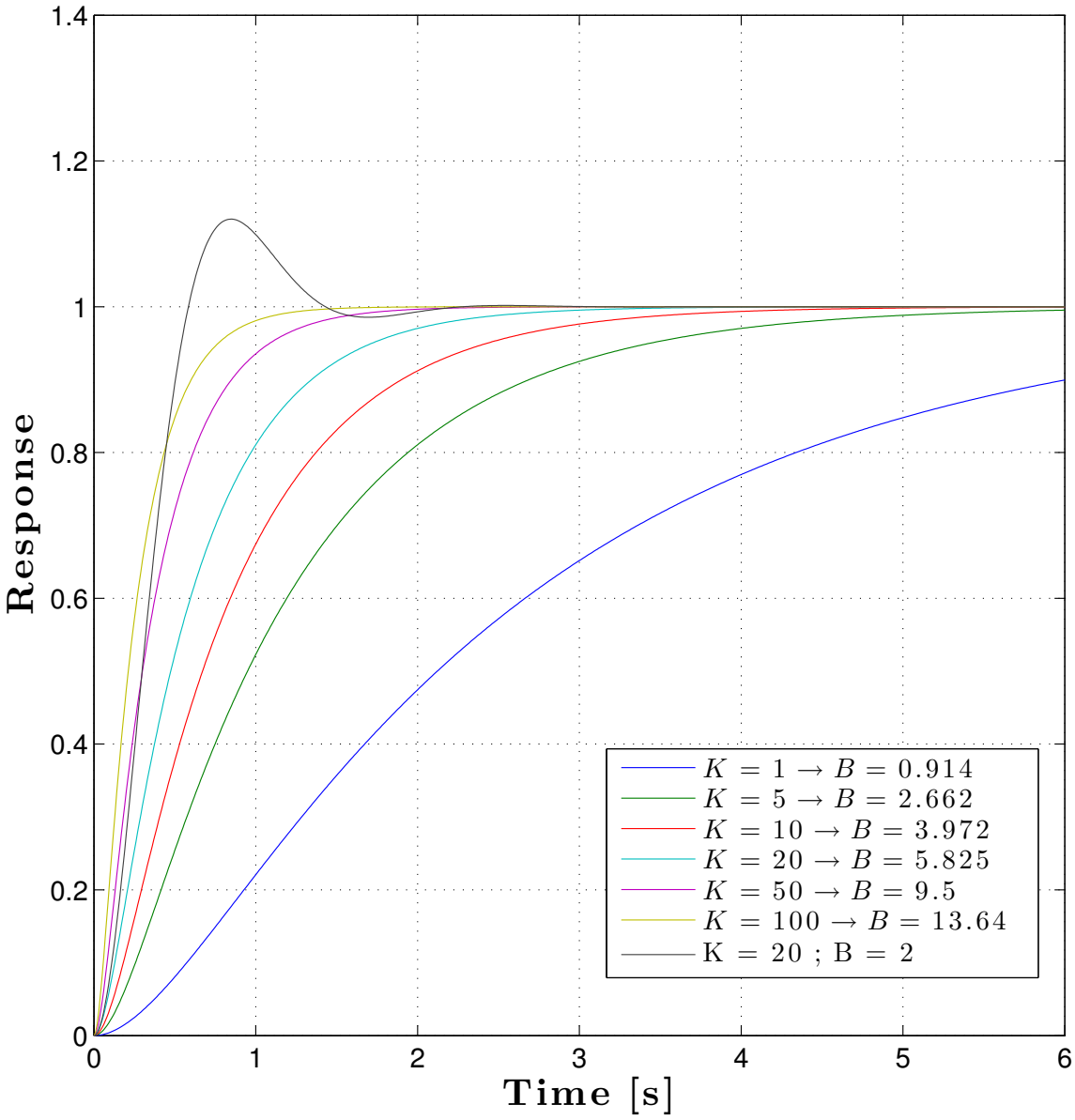

Figure 2.18: Illustrates the response to a unity step input of the system under study with different controller parameters. All but the black plot correspond to a choice of B and K leading to a critically damped system. As it is seen, in the case where the system is not critically damped an overshot and oscillation is present.

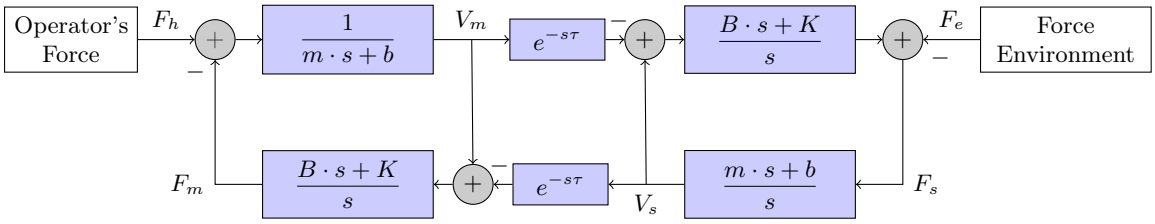

Figure 2.19: Illustrates a bilateral telemanipulation architecture based on the Position-Position Architecture where the physical delay between master and slave has been taken into account.

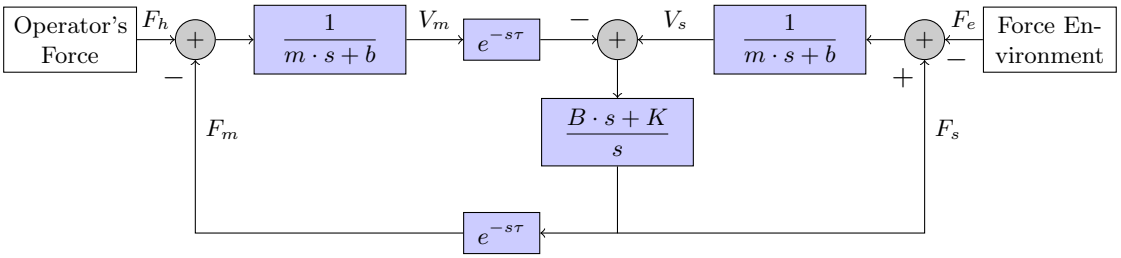

Figure 2.20: Illustrates a bilateral telemanipulation architecture based on the Position-Force Architecture where the physical delay between master and slave has been taken into account.

than the previous cases. Since this thesis focuses on the Position-Force architecture this case will not be further pursued.

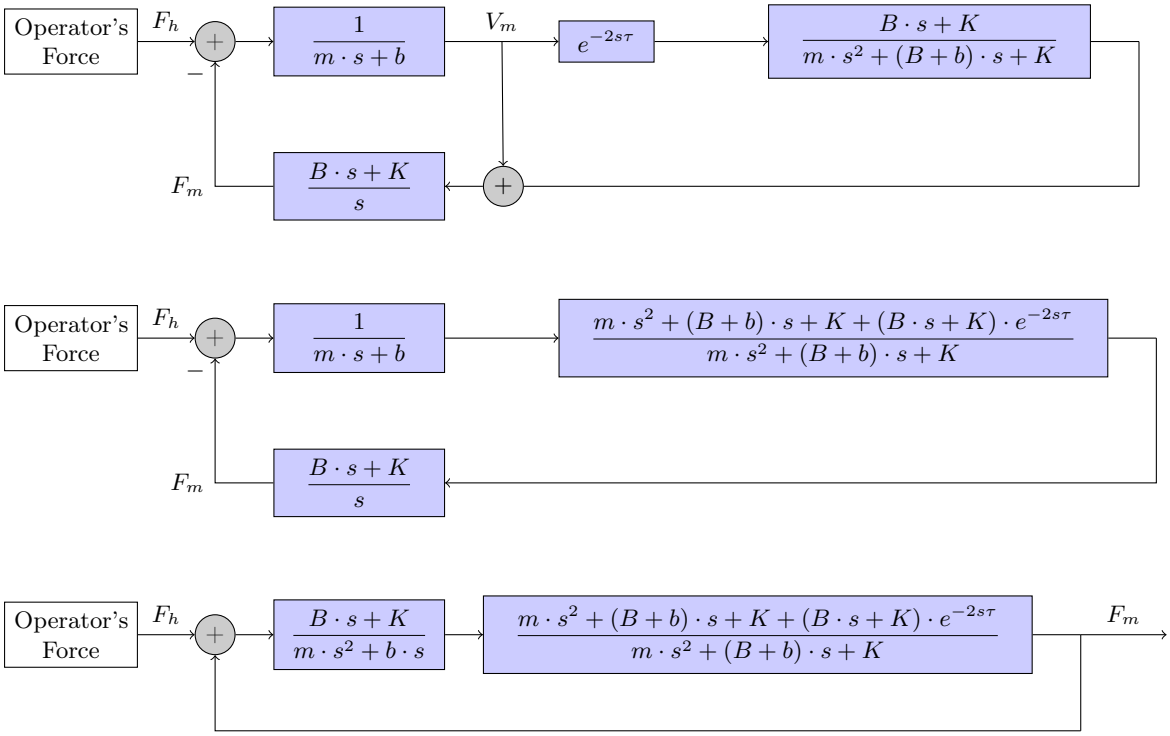

Figure 2.21: Illustrates various steps in the process of simplifying the original Position-Position Symmetric theme into a simple block

As for the Position-Force architecture, the one of greater interest, the simplification is such as seen in Figure **??**.

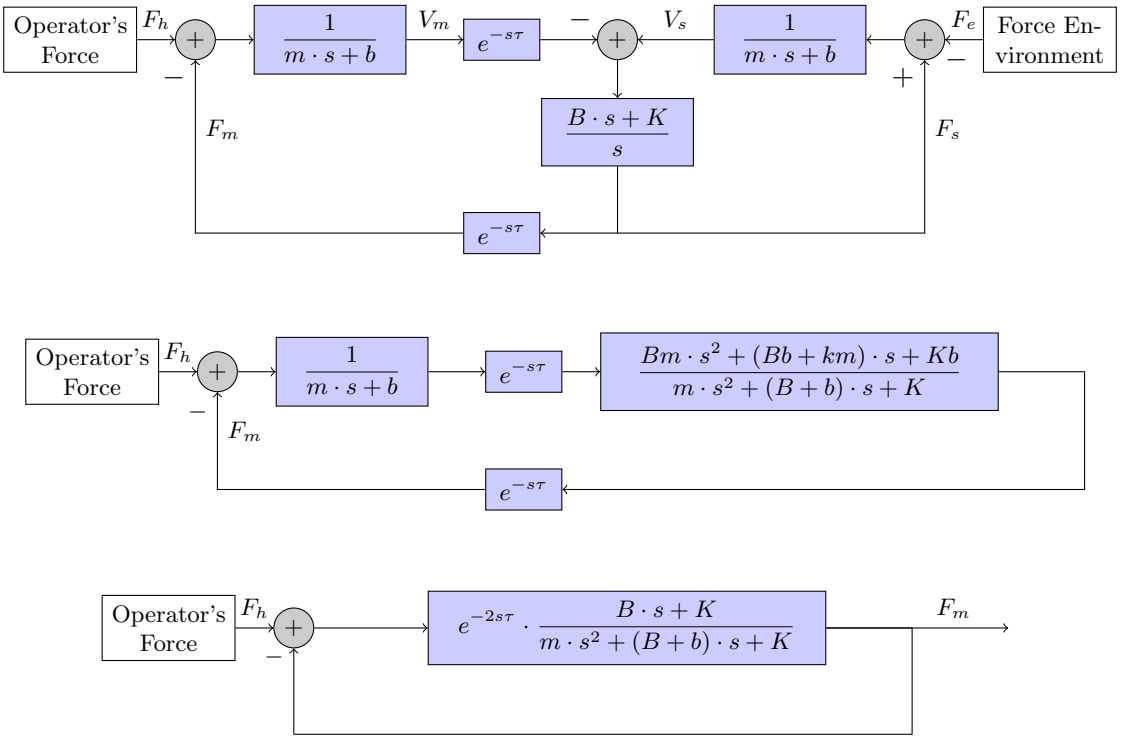

Figure 2.22: Illustrates a bilateral telemanipulation architecture based on the Position-Force Architecture where the different entities have been modeled according to their Laplace transformation

As it can be seen from the final simplification the open-loop expression is no longer a polynomial, making the analysis of the poles and zeros of the system much more involved. Firstly by closing the loop and expanding the complex exponential the result in Equation 2.15 can be reached.

$$T(s) = \frac{(B \cdot s + K) \cdot (cos(2w\tau) + i \cdot sin(2w\tau)) \cdot e^{-\alpha}}{m \cdot s^2 + (2B + b) \cdot s + 2K + (B \cdot s + K) \cdot (cos(2w\tau) + i \cdot sin(2w\tau)) \cdot e^{-\alpha}}. \quad (2.15)$$

The main problem in the simplification procedure is that the variable w, frequency, a sub-group of s, is present both in polynomial form and also within trigonometric functions, this prevents a straight-forward simplification. However if the variable $s = jw + \alpha$ is substituted by $s = jw$ the points at which poles cross over from the negative real par to of the complex plain into the positive real part can be found.

Firstly the functions's denominator is splitted into real and imaginary parts, this is very useful since it is a fact that a pole must have a value of zero both in the real and

imaginary part, so by splitting the expression the variables i can be eliminated. This can be seen in 2.16.

$$T(s) = \frac{N(s)}{D(s)} \quad \rightarrow \quad \begin{cases} \Re\{D(s)\} = K - mw^2 + K\cos(2w\tau) + Bw\sin(2w\tau) \\ \Im\{D(s)\} = bw + Bw \cdot (1 + \cos(2w\tau)) - K\sin(2w\tau) \end{cases} \tag{2.16}$$

If then both these expressions are equated to zero and variables B and K isolated in both cases expressions in terms of w can be found which determine the point at which the system's poles lay exactly in the imaginary axes for a given set of B and K. Such procedure can be seen in Equations 2.17, 2.18, 2.19, and 2.20.

$$\Re\{D(s)\} = 0 = \begin{cases} K = & \dfrac{w\sec(w\tau)^2 \cdot (mw - B\sin(2w\tau))}{2} \\ B = & \dfrac{-(K - mw^2 + K\cos(2w\tau))\csc(2w\tau)}{w} \end{cases} \tag{2.17}$$

$$\Im\{D(s)\} = 0 = \begin{cases} K = & w(b + B(1 + \cos(2w\tau))) \cdot \csc(2w\tau) \\ B = & \dfrac{-b\sec(w\tau)^2}{2} + \dfrac{K\tan(w\tau)}{w} \end{cases} \tag{2.18}$$

$$\left.\begin{array}{l} K = \dfrac{w\sec(w\tau)^2 \cdot (mw - B\sin(2w\tau))}{2} \\ \\ K = w(b + B(1 + \cos(2w\tau))) \cdot \csc(2w\tau) \end{array}\right\} \quad \rightarrow \quad B = \frac{mw\tan(w\tau) - b}{2} \tag{2.19}$$

$$\left.\begin{array}{l} B = \dfrac{-(K - mw^2 + K\cos(2w\tau))\csc(2w\tau)}{w} \\ \\ B = \dfrac{-b\sec(w\tau)^2}{2} + \dfrac{K\tan(w\tau)}{w} \end{array}\right\} \quad \rightarrow \quad K = \frac{mw^2 + bw\tan(w\tau)}{2} \tag{2.20}$$

Now that such expressions have been reached different values of delay, τ, can be fixed and the critical values of the controller's parameters, B and K which render the

system unstable found one in function of the other. Such method has been used to generate Figure 2.23 plotting the critical values of K as a function of B for relatively large delays (up to two hundred) milliseconds and Figure 2.24 for relatively small delays (up to twenty milliseconds).

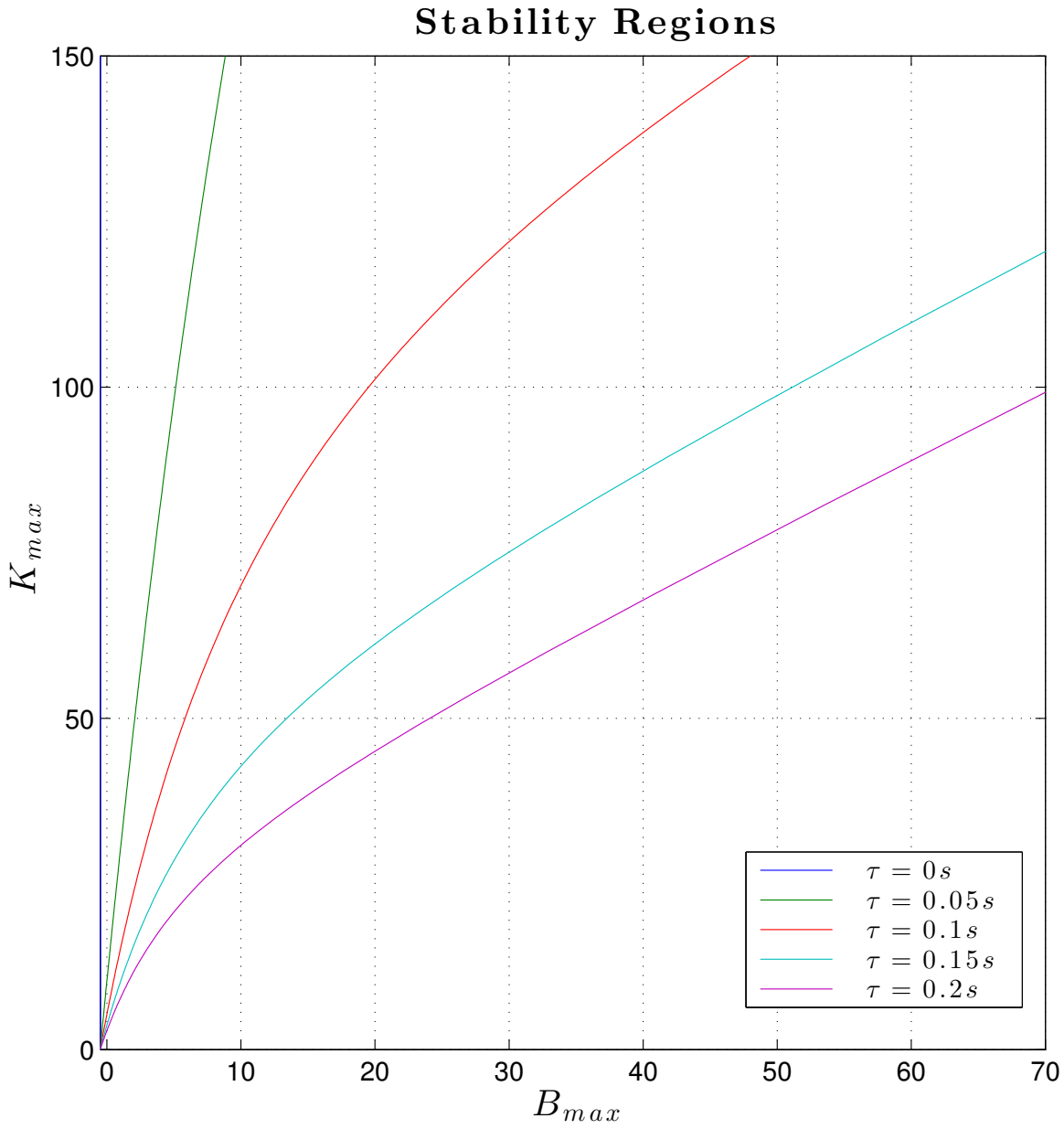

Figure 2.23: This graphs plots the critical values of K in function of B and certain fixed delay, τ for a bilateral system with a delayed transmission where delays are kept smaller than two hundred milliseconds. As in previous cases values for m and b have been chosen to be unity

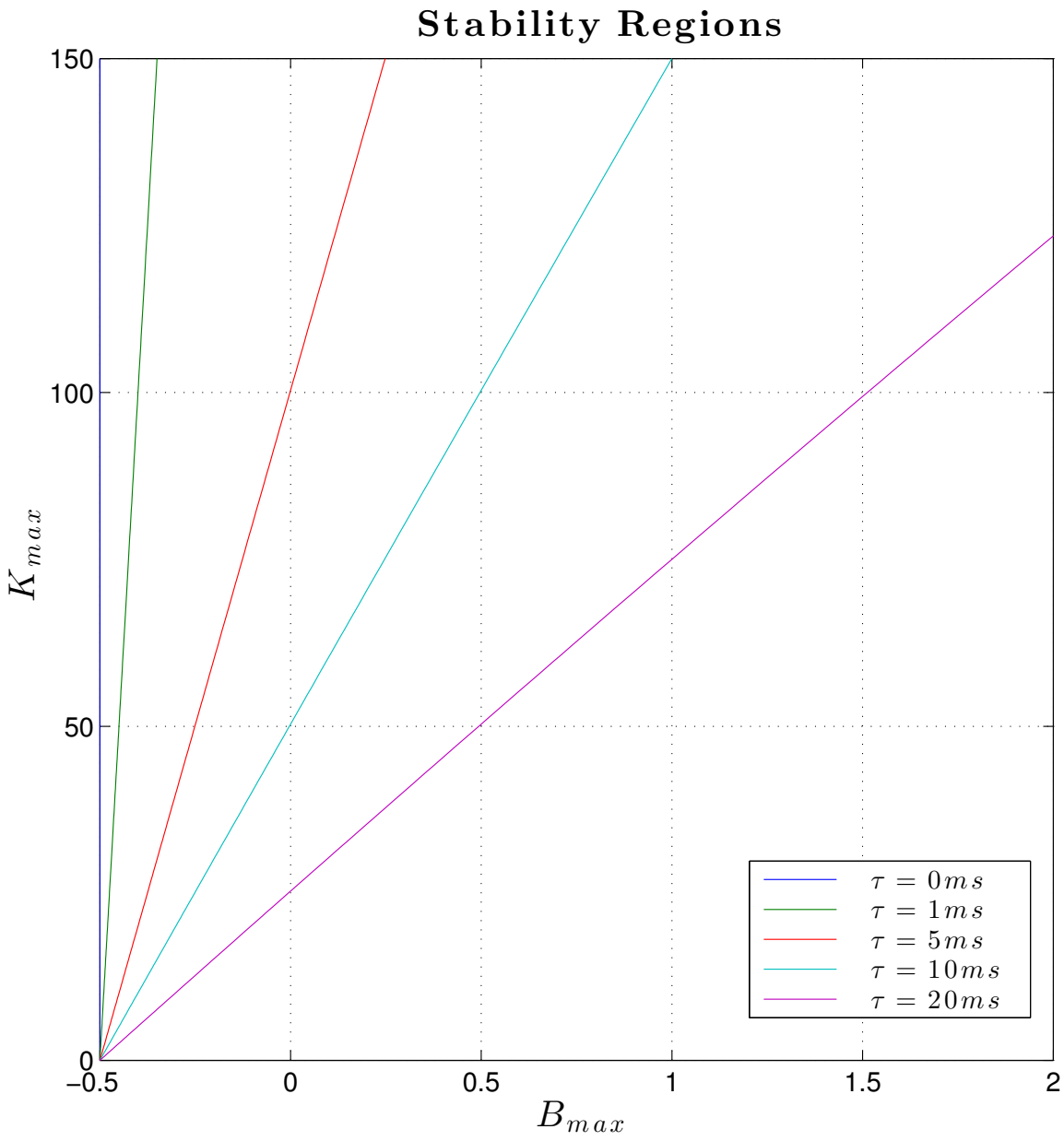

Figure 2.24: This graphs plots the critical values of K in function of B and certain fixed delay, τ for a bilateral system with a delayed transmission where delays are kept smaller than twenty milliseconds.

3

sEMG-Based Arm Impedance Estimation

In this section the way in which the sEMG signals are used to estimate the human limb impedance. As it will be seen a series of assumptions and approximations will be made in order to simplify the analysis, otherwise, a detailed general discussion on the topic could take on multiple thesis like the one at hand and is just not feasible for the study at hand [10].

The most important point behind the whole analysis is to assume (as research supports [7]) that a limb stiffness (in the case at hand the limb being the human arm) depends, at least partially, on the muscular activity of certain muscle along such limb. Having assumed that and taking a series of precautions to make sure that this muscular activity is of importance in the subject's impedance tuning a relationship between the two can be stablished.

3.1 Impedance Characterization

An impedance, as it is understood from a mechanical point of view, is a parameter that describes how a certain body reacts, in terms of motion, to a given force. Therefore, since

mechanical systems are LTI systems, mechanical impedance (from now on referred just as impedance), can be defined in the frequency domain as stated in Equation 3.1,

$$Z(w) = \frac{F(w)}{v(w)} \tag{3.1}$$

where $F(w)$ stands for the force applied (either as a vector or scalar depending on the space at hand) and $v(w)$ as the velocity.

Just like in an electrical impedance (where the term "mechanical impedance" stems from) $Z(w)$ is, in general, a complex quantity in the Laplace space. The relationship between electric and mechanical impedances can be seen if the differential equations leading to both qualities are compared. In Equation 3.2 we can see both these expressions.

$$\text{Electric Impedance} \quad \rightarrow \quad \frac{dV(t)}{dt} = \frac{d^2I(t)}{dt} \cdot L + \frac{dI(t)}{dt} \cdot R + \frac{I(t)}{R} \tag{3.2a}$$

$$\text{Mechanic Impedance} \quad \rightarrow \quad \frac{dF(t)}{dt} = \frac{d^2v(t)}{dt} \cdot m + \frac{dv(t)}{dt} \cdot b + v(t) \cdot K \tag{3.2b}$$

As it can be seen if mechanical forces are regarded as potential functions and mechanical velocities as generic flow functions both equations describe a second order differential system where only the constants take on different values.

If this expressions are then expressed in the Laplace space and the corresponding expressions for the impedances isolated the result shown in Equation 3.3.

$$\text{Electric Impedance} \quad \rightarrow \quad \frac{V(s)}{I(s)} = \frac{L \cdot s^2 + R \cdot s + 1/C}{s} \tag{3.3a}$$

$$\text{Mechanic Impedance} \quad \rightarrow \quad \frac{F(s)}{v(s)} = \frac{m \cdot s^2 + b \cdot s + K}{s} \tag{3.3b}$$

This analogy between this two type of systems means that all the theory developed for electric systems can be effortless ported to mechanical systems and vice-versa. All theorems such as Thevening and Norton equivalents, and of course all control theory being clear examples of such analogy. In Figure 3.1 a particular example (in fact the generic system described in Equations 3.2) of the application of this analogy can be seen. As the

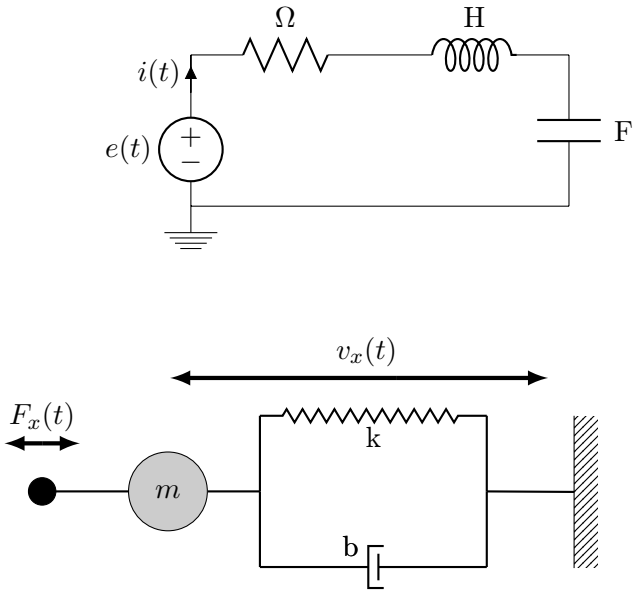

Figure 3.1: Illustrates the analogy between a second order electronic system and a second order mechanic system. In the electric circuit $e(t)$ is the potential function and $i(t)$ is the flow function while in the mechanical system $F_x(t)$ is the potential functions (the force applied) and $v_x(t)$ the flow functions (the velocity of the system).

previous equations indicate the differential equations governing both systems in such figure are of the same form.

It is of some importance to establish the correspondence between both systems' parameters.

Table 3.1: Here the correspondence between the two system's parameters can be seen.

Mechanical System		**Electric System**
m	$\rightarrow$	L
b	$\rightarrow$	R
k	$\rightarrow$	$^1/_C$

By taking a closer look at this correspondences it can be seen that, as stated before, the mechanical damper with factor $b \cdot [^{Nm}/_s]$ corresponds to the resistor in the electric circuit. This supports the previous statement that in a mechanical system de damper is the only element where energy can be lost (either through heat, noise, mechanical deformations, etc.) while all other elements only store and release potential and kinetic energy respectively.

With all this theory that has been introduced it becomes clear that to fully characterize a mechanical impedance three parameters must be known, mass m, damping factor b, and compliance value k. However the aim of this thesis is only to tune the bilateral controller to match the stiffness (or compliance) of the human arm. A little confusion

could arouse at this point concerning the terms used, the controller being tuned is called an impedance controller (even though it has a mass parameter of zero), however, it is tuned only as far as it's stiffness go. Also, as it will be seen later, some concerns arouse when characterizing and measuring the operator's limb stiffness.

3.2 Stiffness Estimation

In order to correlate the sEMG readings to the limb's stiffness an initial value of such stiffness is needed. The most reliable and widely accepted methods for doing so rely on applying perturbations to the endpoint, either perturbations of a constant force and then measuring the corresponding motion of the limb or of a constant displacement and then measuring the resulting force.

A very important fact about human limb stiffness is that it's final value is a result of combining two different factors, what is called *geometric stiffness* and what is called *joint stiffness*. These two factors isolate the main two variables that impact the endpoint stiffness, which are the geometrical configuration of the limb, and the stiffness of each joint along the limb. As a clear example of the combination of these two factors one could imagine a tennis player, while such subject could stiffen up the endpoint (in this case considering such as his/her hand) a lot if he/she was hitting the ball with his elbow folded and the racket close to his/her chest it would take a lot more agonist-antagonistic force along the limb to be able to present the same stiffness with ones arm extended and far away from the body. It is in fact the geometric stiffness which has been proved to play a greater roll in stiffness tuning while joint stiffness has only a limited impact on the final endpoint stiffness [19].

Such principle is depicted in Figure 3.2 where ellipses depict the endpoint stiffness, the distance from this last one to the edge of the ellipse being proportional to the stiffness in the bi-dimensional plane under analysis. As it is shown in the figure with the same agonist-antagonistic muscle activation a subject's endpoint stiffness is greater is different directions depending on limb's geometrical position.

Since this thesis in concerned only with the relationship between antagonist muscle activation and endpoint stiffness the task used throughout should be constricted to a reduced work area to be able to assume that geometrical stiffness variations are kept to a minimum.

The selected task, because of the mentioned reasons, was kept very simple and consisted of a one DOF (Degree Of Freedom) joystick as Figure 3.3 shows, able to apply and measure torques in both directions with a resolution of twelve bits and a bipolar full scale value or 1.7 Nm, the step or LSB (Least Significant Value) being $1.7/2^{12}$ $[Nm]$ and

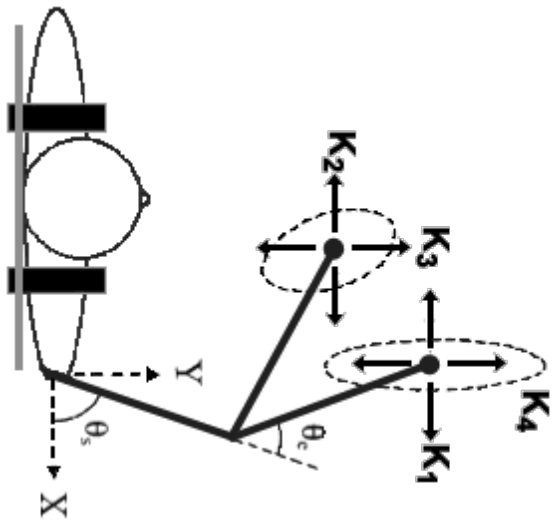

Figure 3.2: Illustrates the influence of a limb's geometrical position in the final endpoint stiffness.

capable of measuring positions with a resolutions of ten bits, the LSB or resolution being $^{2\pi}/_{2^{10}}$ $[rads]$.

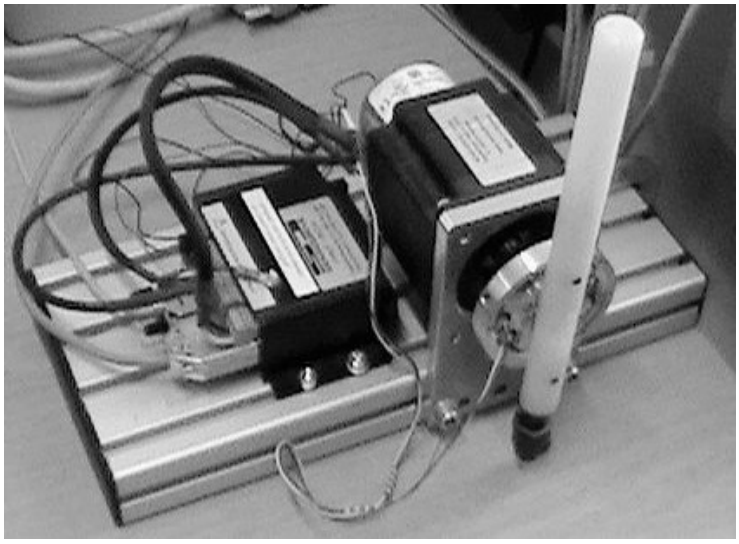

Figure 3.3: Illustrates the influence of a limb's geometrical position in the final endpoint stiffness.

In order to be able to induce different stiffness a set of tasks were designed. The subject under test would be ask to reach a certain position with the manipulandum within a certain tolerance while a disturbing force field was applied. Experience showed that a suitable disturbing force field could be one were a torque proportional to the velocity of the joystick was applied in it's same direction, as expressed in Equation 3.4 where τ stands for the torque applied, ϕ for a gain parameter greater than zero, and w for the angular velocity. Acceptable parameters of ϕ were found to be [0.05, 0.075, 0.1, 0.125, 0.15, 0.2, 0.25, 0.3] $Nm/(^{rad}/_{s})$.

$$\tau(t) = \phi \cdot w(t) \tag{3.4}$$

Each subject was then told to reach a series of alternating different positions, which were restricted only to -0.5 rad, 0 rad, and 0.5 rad and once the subject had managed to maintain the manipulandum within the tolerance for a random period of time ranging from one to two seconds a torque perturbation in a random direction was applied. This perturbation consisted in a $\pm 0.5Nm$ torque lasting $300ms$. This procedure was repeated for all values of ϕ to certify that such force field was capable of inducing different limb's stiffnesses. It was made very clear to all subjects that the exercise had to be performed with the lowest stiffness possible, this is a very important factor as a subject could perfectly perform all rounds of the experiment with a very high stiffness since it is totally possible to perform the rounds with a low intensity unstable force field with a high limb stiffness.

Certain aspects need to be considered when defining such perturbation and when reading the corresponding results. On one hand the perturbation (or the point within such where the stiffness reading is extracted from) needs to be short enough in order to don't allow human reflexes to affect the reading. On the oder hand the perturbation needs to be strong enough so the point of stability is reached within the aforementioned window, this being a problem mainly caused by the associated masses and dampings present in the system. However, specially when measuring low stiffness it is crucial that the perturbation is not too strong as it must not displace the human limb excessively as that would produce important variations in the geometric stiffness of the limb.

Under the defined situations, in an ideal case scenario in which neither the robot nor the human arm had any damping or mass so only a spring was being measured the expected waveform would be one as shown in Figure 3.4.

However, because of the multiple reasons mentioned the response from a real test is far from ideal, an example of such can be seen in Figure 3.5. It was stablished that the increment in position that would actually be accounted for would be the difference between the position at the start of the perturbation and that where the first local minimum or maximum occurs, in the figure such points are marked.

Even greater concerns appear whenever the measured stiffness is low enough so that a point of equilibrium (local minimum or maximum) is not reached within the perturbation window. In such cases the algorithm used considered the local minimum or maximum as the value of the position at the end of the window. While this is still a measurement that should be somehow correlated to the subject's stiffness it is not by far precise nor reliable. Because of restrictions imposed by the hardware used these effects could not be avoided.

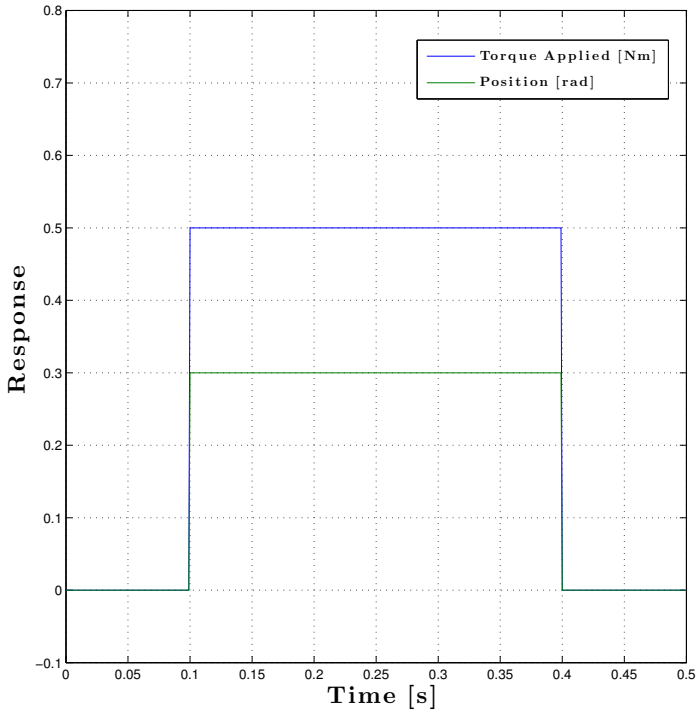

Figure 3.4: Illustrates the results that would be expected in an ideal scenario where neither the operator's limb nor the actuator have any mass and where the operator's limb behaves exactly as a spring.

All in all each experiment lasted about 60 minutes and no subject reported discomfort, fatigue and/or pain. A total of 6 able-bodied subjects (age 25.8 ś 1.8yrs, min 23, max 28) joined the experiment.

As mentioned the subject's arm stiffness was measured by evaluating the ratio between the amount of the torque perturbation and the angular displacement the manipulandum had undergone within an interval t_0, t_E,

$$k_h = \frac{\tau(t_E) - \tau(t_0)}{\theta(t_E) - \theta(t_0)} = \frac{\Delta\tau}{\Delta\theta}. \tag{3.5}$$

where t_E, beeing $t_p > t_E > t_0$, is the time at which the first extremum value of θ would be found. The amount of the perturbation was evaluated using the internal torque sensor, rather than fixing the value at the commanded value of 0.5Nm. The sensed value would actually match 0.5Nm within $\pm$0.01Nm.

A linear fit was employed to check whether a linear relationship would hold between the strength of the torque field and the measured arm stiffness as of Equation 3.6:

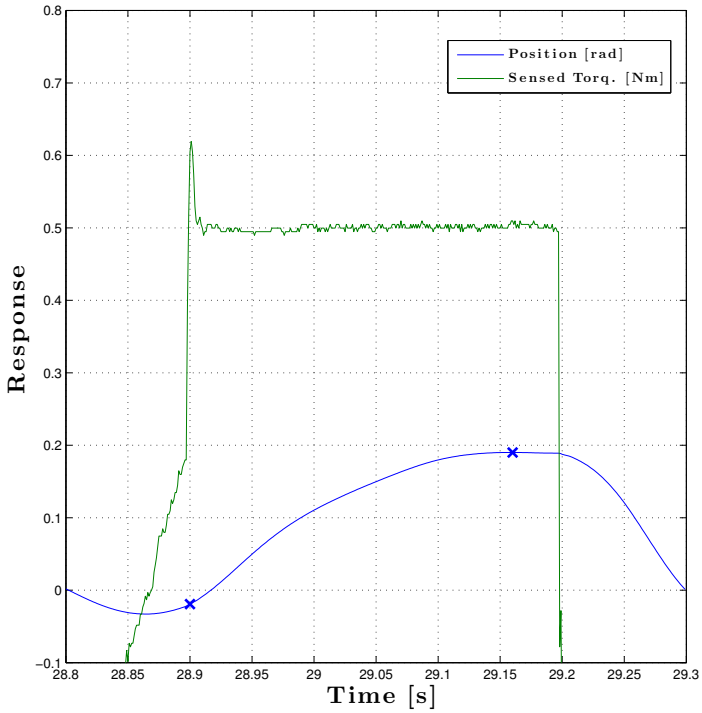

Figure 3.5: Illustrates a real result from a perturbation-based stiffness measurement where a delay in present due to the presence of various masses which in turn create inertial forces and where also other factors concerning the operator's limb com into effect.

$$k_h = \alpha' \phi + \beta' \tag{3.6}$$

It turns out that this is the case uniformly for all subjects. The r-squared coefficients of k_h linear fit are reported in Table 3.2 for each subject where it is clear that the relationship between the strength of the unstable force field and the measurement obtained is clearly linear.

Table 3.2: R-squared coefficients of the linear fits of k_h (Eqs. 3.5)

	s#1	**s#2**	**s#3**	**s#4**	**s#5**	**s#6**	**mean ± std.**
R^2 coefficient	0.963	0.977	0.718	0.952	0.893	0.873	0.896 ± 0.096

3.3 sEMG Based Arm Stiffness Estimation

Now that a method for inducing and measuring variables values of limb stiffness has been developed it is time to correlate this data to muscle activation.

In order to gather the desired sEMG data a total of ten surface differential electrodes were used, for more information on those please see Annex D. The position of the electrodes was initially decided in order to try to capture the co-activation of antagonistic muscles, however, later processing (detailed later) was performed in order to obtain an optimum reading. The positions of the electrodes can be seen in Figure 3.6.

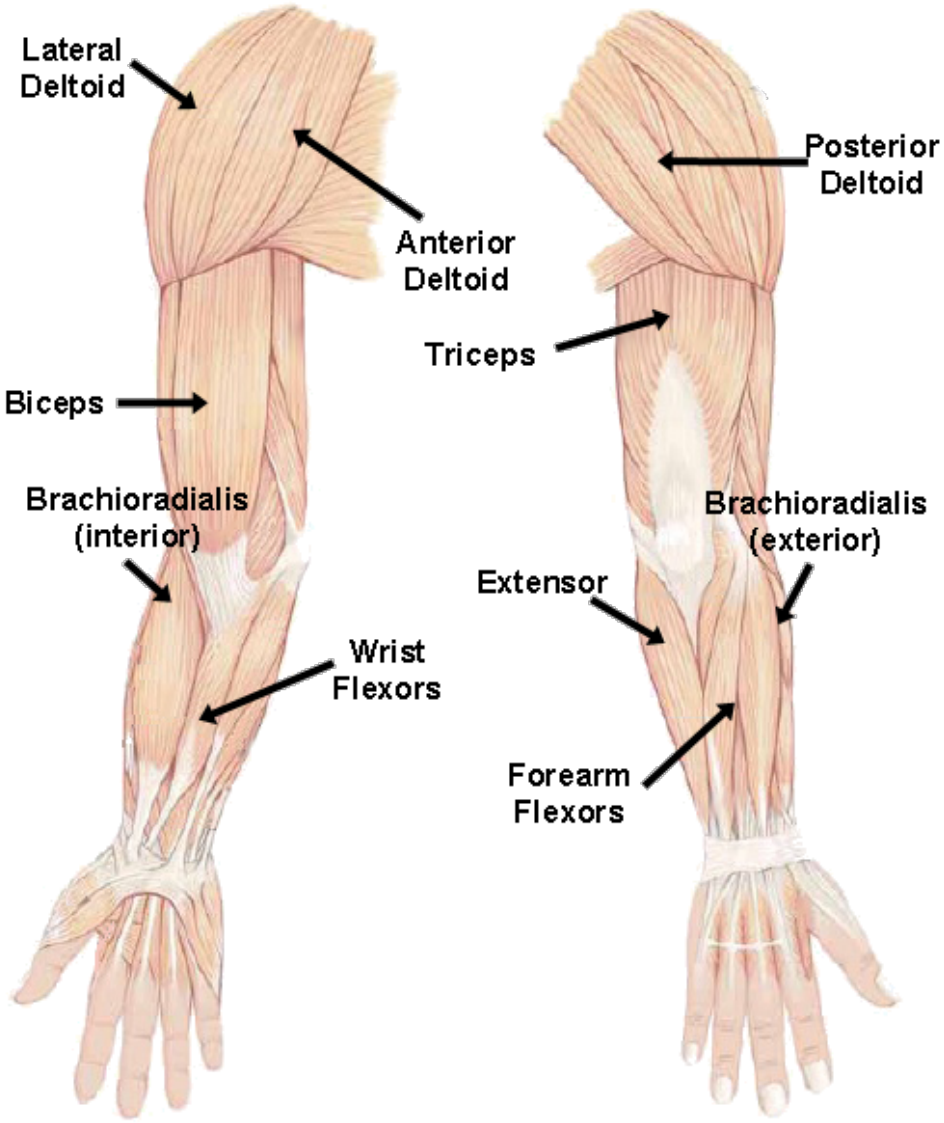

Figure 3.6: Illustrates the approximated position of the ten electrodes that were used throughout the sEMG readings.

At this point a certain issue rises. While a measurement of the stiffness for every repetition of the experiment is known, ten, not one, sEMG electrodes are placed on the subject's limb. Since the only information that is to be extracted from all this data is a reading proportional to overall muscle activation (there was no constant force field at the time of the measurement so it is assumed that the only muscle activation is that which is responsible for stiffness tuning through agonistic-antagonistic muscle activation) a dimensionality reduction method was applied to the sEMG data.

Such method was chosen to be Principal Component Analysis (PCA from now on) which works by finding out the combination of variables leading to a greater output variance, averaging those variables that show a high correlation. Mathematically this is achieved by obtaining the Eigenvectors of the covariance matrix, up to this point, no dimensionality reduction or loss of data has happened yet as all that this does is express the data in an another vector coordinate system. However, if the Eigenvalues of the resulting vectors are evaluated the importance of each Eigenvector at representing the covariance matrix can be evaluated, if the eigenvectors with the lowest Eigenvalue are eliminated from the base a lower loss of variance (and therefore entropy) will occur.

As an example of PCA imagine a set of bi-dimensional data such as the one depicted in Figure 3.7 where it can be observed there is a certain correlation of variables x and y.

If the data conforming the graph is expressed in two row vectors $\vec{x_1}$ and $\vec{x_2}$ the covariance matrix is defined as the matrix whose (i, j) entry is the covariance of $\vec{x_i}$ and $\vec{x_j}$,

$$\Sigma_{ij} = \mathrm{cov}(\vec{x_i}, \vec{x_j}) = \mathrm{E}\left[(\vec{x_i} - \mu_i)(X\vec{x_j} - \mu_j)\right] \tag{3.7}$$

where μ_n is the expected value (or mean) of each variable over the whole set, the general expression for the covariance matrix being:

$$\Sigma = \begin{bmatrix} \mathrm{E}[(\vec{x_1} - \mu_1)(\vec{x_1} - \mu_1)] & \mathrm{E}[(\vec{x_1} - \mu_1)(\vec{x_2} - \mu_2)] & \cdots & \mathrm{E}[(\vec{x_1} - \mu_1)(\vec{x_n} - \mu_n)] \\ \mathrm{E}[(\vec{x_2} - \mu_2)(\vec{x_1} - \mu_1)] & \mathrm{E}[(\vec{x_2} - \mu_2)(\vec{x_2} - \mu_2)] & \cdots & \mathrm{E}[(\vec{x_2} - \mu_2)(\vec{x_n} - \mu_n)] \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{E}[(\vec{x_n} - \mu_n)(\vec{x_1} - \mu_1)] & \mathrm{E}[(\vec{x_n} - \mu_n)(\vec{x_2} - \mu_2)] & \cdots & \mathrm{E}[(\vec{x_n} - \mu_n)(\vec{x_n} - \mu_n)] \end{bmatrix} \tag{3.8}$$

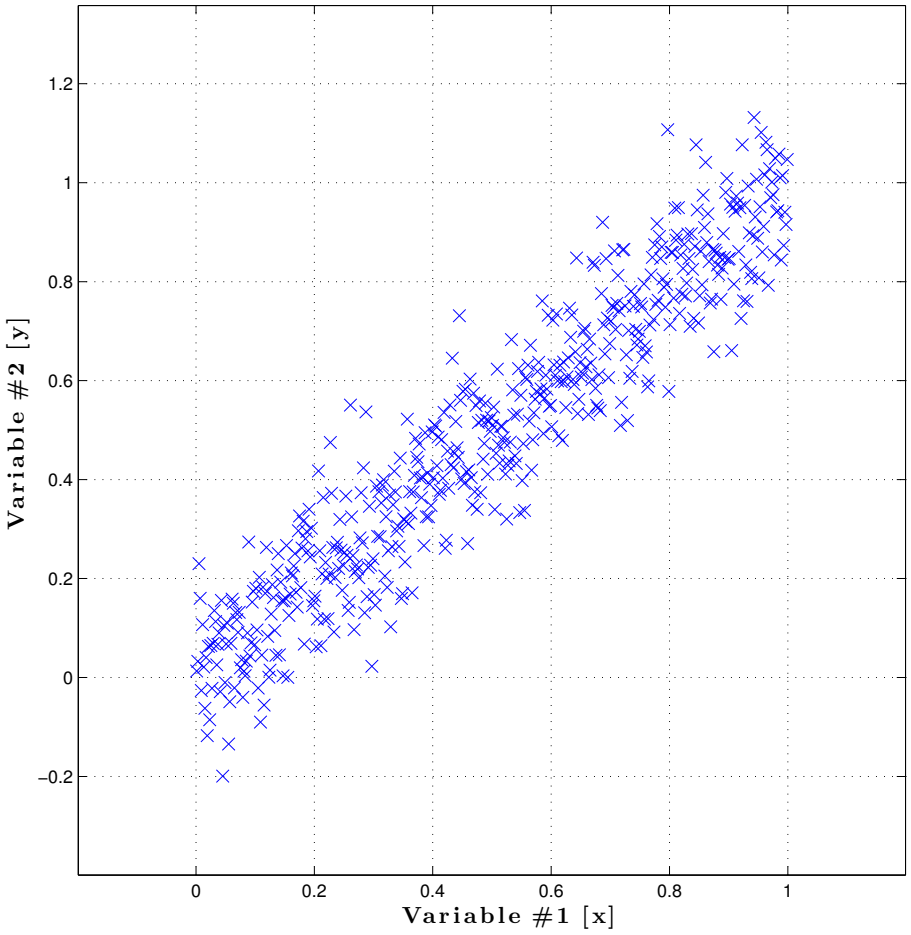

Figure 3.7: Illustrates the approximated position of the ten electrodes that were used throughout the sEMG readings.

In the example at hand, where the data has been generated by adding white gaussian noise of -20 dBW of average power to two initially identical variables, the covariance matrix is:

$$\Sigma = \begin{bmatrix} \mathrm{E}[(\vec{x_1} - \mu_1)(\vec{x_1} - \mu_1)] & \mathrm{E}[(\vec{x_1} - \mu_1)(\vec{x_2} - \mu_2)] \\ \\ \mathrm{E}[(\vec{x_2} - \mu_2)(\vec{x_1} - \mu_1)] & \mathrm{E}[(\vec{x_2} - \mu_2)(\vec{x_2} - \mu_2)] \end{bmatrix} = \begin{bmatrix} 0.0835 & 0.0845 \\ 0.0845 & 0.0966 \end{bmatrix} \tag{3.9}$$

PCA then relays on the Eigenvectors and Eigenvalues of such matrix in order to be able to put into perspective the dimensions of the data where more variance is present. The definition of Eigenvectors and Eigenvalues can be found in Equation 3.10:

$$\begin{bmatrix} COV. \\ MATRIX \\ n \cdot n \end{bmatrix} \cdot \begin{bmatrix} e.\vec{v}_1 & e.\vec{v}_2 & e.\vec{v}_3 & \cdots & e.\vec{v}_4 \end{bmatrix}$$

$$= \begin{bmatrix} e.val(v_1) & e.val(v_2) & e.val(v_3) & \cdots & eval(v_n) \end{bmatrix} \cdot \begin{bmatrix} e.\vec{v}_1 \\ e.\vec{v}_2 \\ e.\vec{v}_3 \\ \vdots \\ e.\vec{v}_n \end{bmatrix} \tag{3.10}$$

The Eigenvectors are basically vectors orthogonal to the base defined by the covariance matrix and the Eigenvalues are their relative weight in the new base. In the present example, the Eigenvectors and their respective Eigenvalues turn out to be:

$$Eigenvectors = \begin{bmatrix} e.\vec{v}_1 & e.\vec{v}_2 \end{bmatrix} = \begin{bmatrix} -0.7339 & 0.6793 \\ 0.6793 & 0.7339 \end{bmatrix}$$

$$Eigenvalues = \begin{bmatrix} e.val(v_1) & e.val(v_2) \end{bmatrix} = \begin{bmatrix} 0.0052 & 0 \\ 0 & 0.1748 \end{bmatrix} \tag{3.11}$$

As it can be seen the second Eigenvalue is much higher than the first one, indicating that the second Eigenvector (the second column vector) contains much more variance than the first one. Notice that this Eigenvector, the second one, is the almost exactly the diagonal around where the data was generated from: $\arctan(0.7339/0.6793) \approx 45°$. This data, the relative values of the different Eigenvalues is what is used to decide what dimension of the new base are less "important" and can be therefore eliminated causing the least loss of entropy. By transforming the original data set to the new space (the second Eigenvector being the conversion matrix) the dimensionality of the original data would be reduced to only one dimension and the loss of data would be kept to a minimum.

In order to try to correlate the sEMG data to the intensity of the unstable force field (and therefore to the measured stiffness) the values of all sEMG electrodes were averaged uniformly over the time of one second previous to the perturbation measurement, by doing this one reading from each electrode is available per repetition just like one perturbation-based stiffness measurement is obtained for each. Results showed that in all cases the greatest Eigenvalue was at least ten times greater than the second, justifying the dimen-

sionality reduction and indicating that the subjects under test were tuning joint stiffness by general co-activation of agonist-antagonistic muscles during the whole exercise.

At that point, and just like it had been done with the correlation between the unstable force field and the perturbation-based stiffness measurements, the resulting sEMG readings were linearly fitted to the intensity of the unstable force field as shown in Equation 3.12, where a is the result of the dimensionality reduction or from now on "muscle activation factor". The results of this fit can be seen in Table 3.3 and as indicated by the R^2 coefficients the muscle activation is almost totally linearly related to the intensity of the field, just like the measured stiffness (k_h) has previously been found to be.

$$a = \alpha'' \phi + \beta'' \tag{3.12}$$

Table 3.3: R-squared coefficients of the linear fits of a (Eqs. 3.12)

	s#1	**s#2**	**s#3**	**s#4**	**s#5**	**s#6**	**mean ± std.**
R^2 coefficient	0.896	0.965	0.960	0.938	0.877	0.850	0.914 ± 0.047

Since both the stiffness measured and the muscle activation are linearly related to the strength of the unstable force field applied it is obvious that they will both be linearly related too. If Equations 3.6 and 3.12 are combined a direct relationship between the muscle activation factor and the estimated stiffness can be extracted, this is shown in Equation 3.13.

$$\overline{k_h} = \frac{\alpha'}{\alpha''} a + (\beta' - \frac{\alpha' \beta''}{\alpha''}) = \alpha''' a + \beta''' \tag{3.13}$$

3.4 Experimental Results

Once the method for estimating the limb stiffness from the sEMG data had been developed a total of a total of 6 able-bodied subjects (age 25.8 ś 1.8yrs, min 23, max 28) joined the experiment. The sEMG data was gathered and saved while the same exercices that had been conducted in order to gather the initial perturbation-based stiffness measurements were conducted. Again it was made very clear to all subjects that the exercise had to be performed with the lowest stiffness possible, this is a very important factor as a subject

could perfectly perform all rounds of the experiment with a very high stiffness since it is totally possible to perform the rounds with a low intensity unstable force field with a high limb stiffness.

After the experiment had been completed the whole set was used to extract the linear fitting values for Equation 4.3 and then the error between the estimated stiffness based on the sEMG values versus the perturbation-based ones was evaluated. Table 3.4 shows the Root Means Squared Error between those two values is shown.

$$\overline{k_h} = \frac{\alpha'}{\alpha''}a + (\beta' - \frac{\alpha'\beta''}{\alpha''}) = \alpha'''a + \beta''' \tag{3.14}$$

Table 3.4: Root-Mean-Square error of the perturbation-based measured stiffness (Eq. 3.5 vs. sEMG-estimated k_h (Eq. 4.3) for each subject.

	s#1	**s#2**	**s#3**	**s#4**	**s#5**	**s#6**	**mean ± std.**
rMSE$[\frac{Nm}{rad}]$	0.449	0.204	0.471	0.372	0.068	0.293	0.31 ± 0.154

Next graphs showing the results for the six subjects are shown. Three graphs are shown for each subject, the first two showing the linear fits of the perturbation-based measured stiffness and muscle activation versus the intensity of the unstable force field. The third one shows both the perturbation measured stiffness and the sEMG-based estimated stiffness and the rMSE between the two.

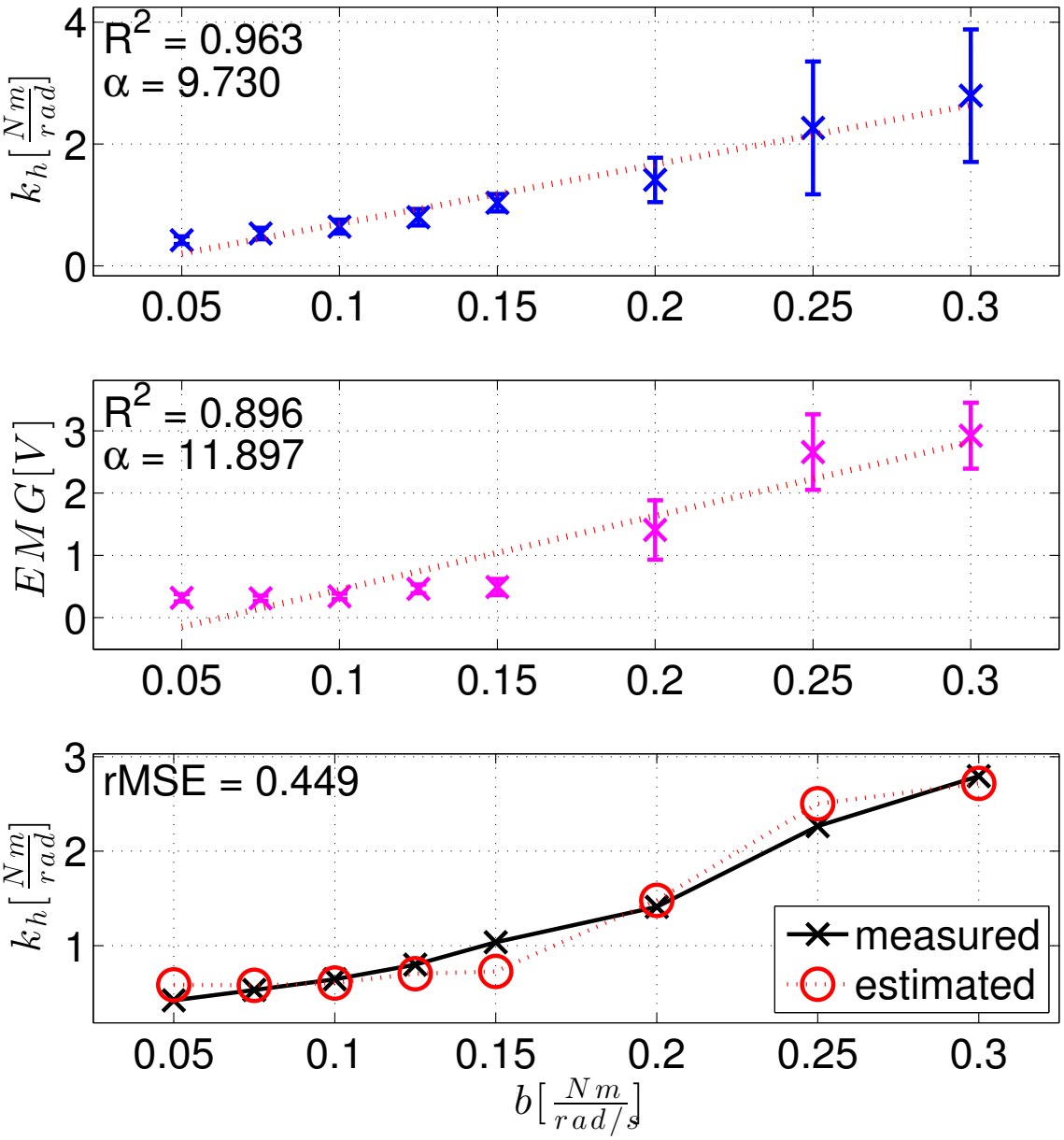

Figure 3.8: Illustrates the experimental results for subject #1. The first two graphs depict the linear fits of the relation between the intensity of the unstable force field and the measured stiffness and muscle activation respectively. The third graph depicts the discrepancy between the perturbation-based stiffness measurements and the sEMG-Based estimations.

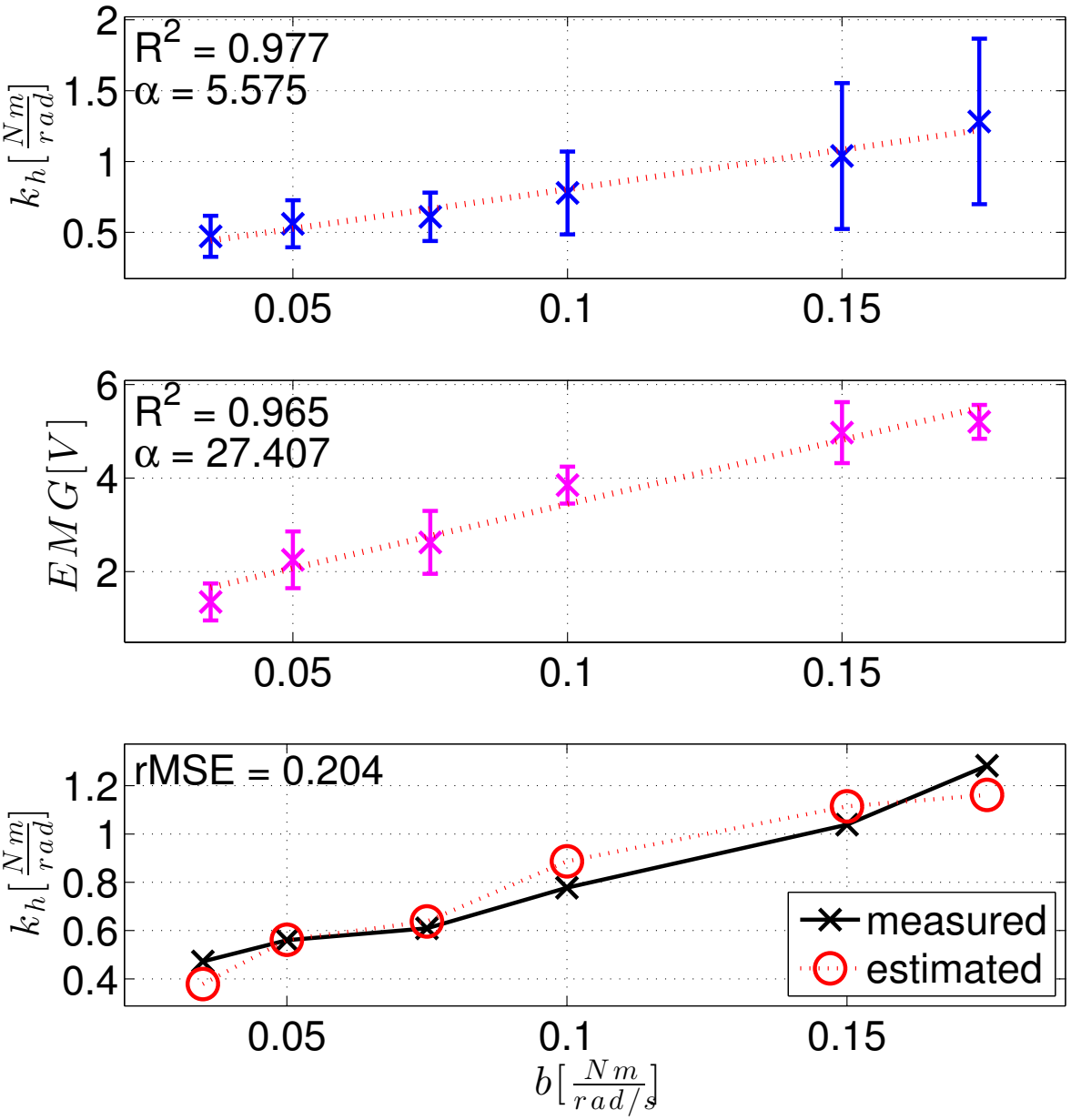

Figure 3.9: Illustrates the experimental results for subject #2. The first two graphs depict the linear fits of the relation between the intensity of the unstable force field and the measured stiffness and muscle activation respectively. The third graph depicts the discrepancy between the perturbation-based stiffness measurements and the sEMG-Based estimations.

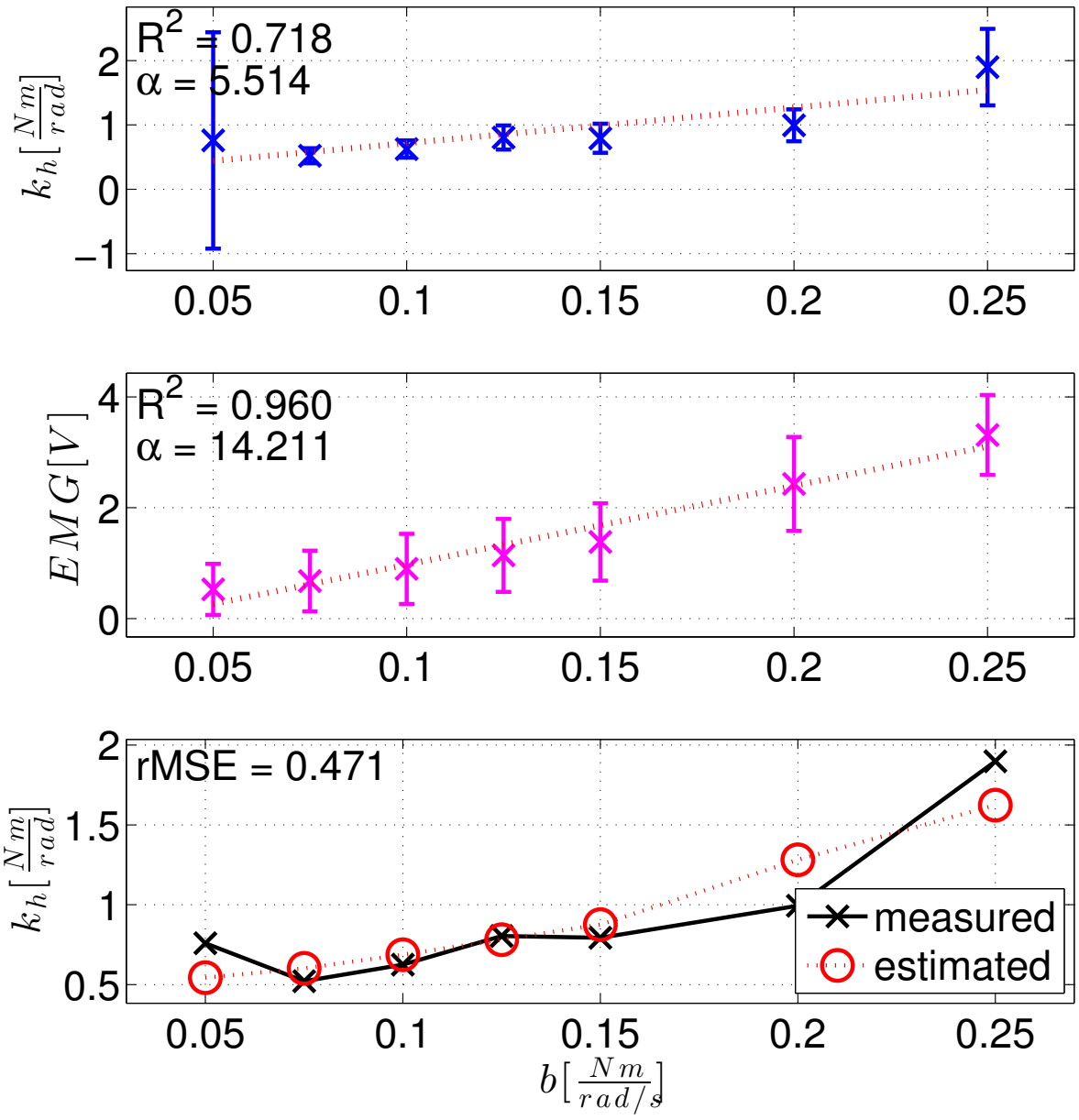

Figure 3.10: Illustrates the experimental results for subject #3. The first two graphs depict the linear fits of the relation between the intensity of the unstable force field and the measured stiffness and muscle activation respectively. The third graph depicts the discrepancy between the perturbation-based stiffness measurements and the sEMG-Based estimations.

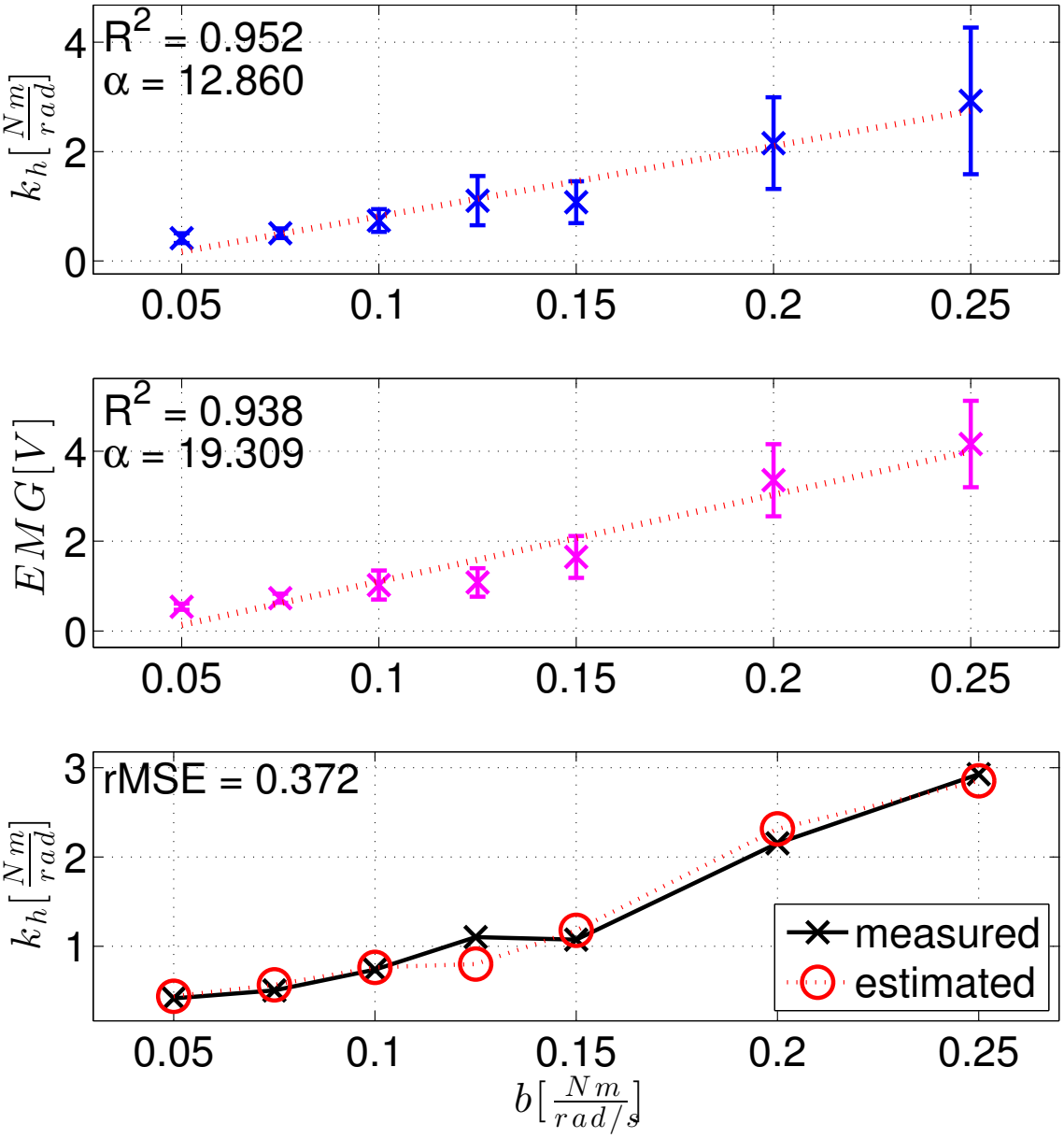

Figure 3.11: Illustrates the experimental results for subject #4. The first two graphs depict the linear fits of the relation between the intensity of the unstable force field and the measured stiffness and muscle activation respectively. The third graph depicts the discrepancy between the perturbation-based stiffness measurements and the sEMG-Based estimations.

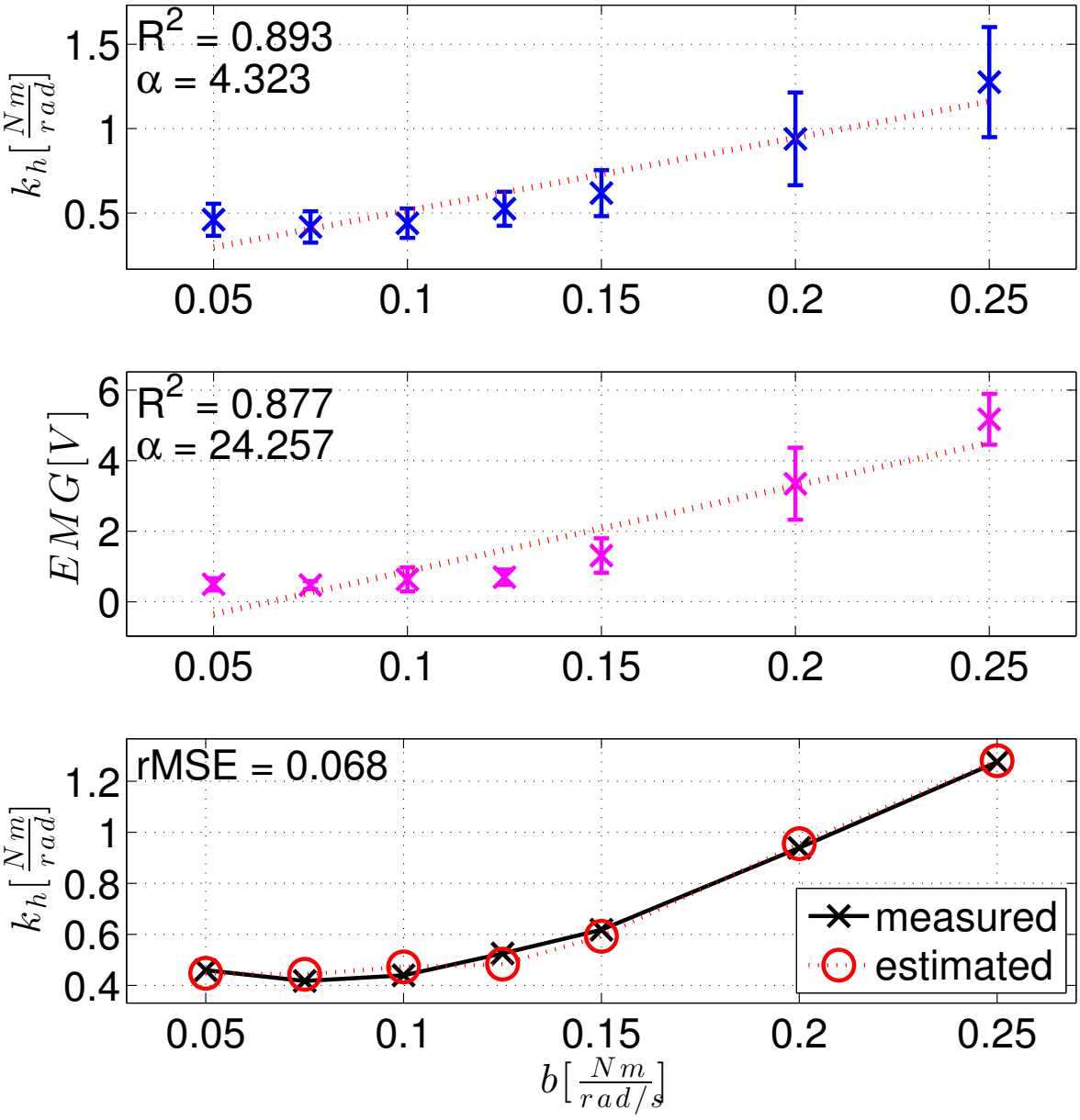

Figure 3.12: Illustrates the experimental results for subject #5. The first two graphs depict the linear fits of the relation between the intensity of the unstable force field and the measured stiffness and muscle activation respectively. The third graph depicts the discrepancy between the perturbation-based stiffness measurements and the sEMG-Bases estimations.

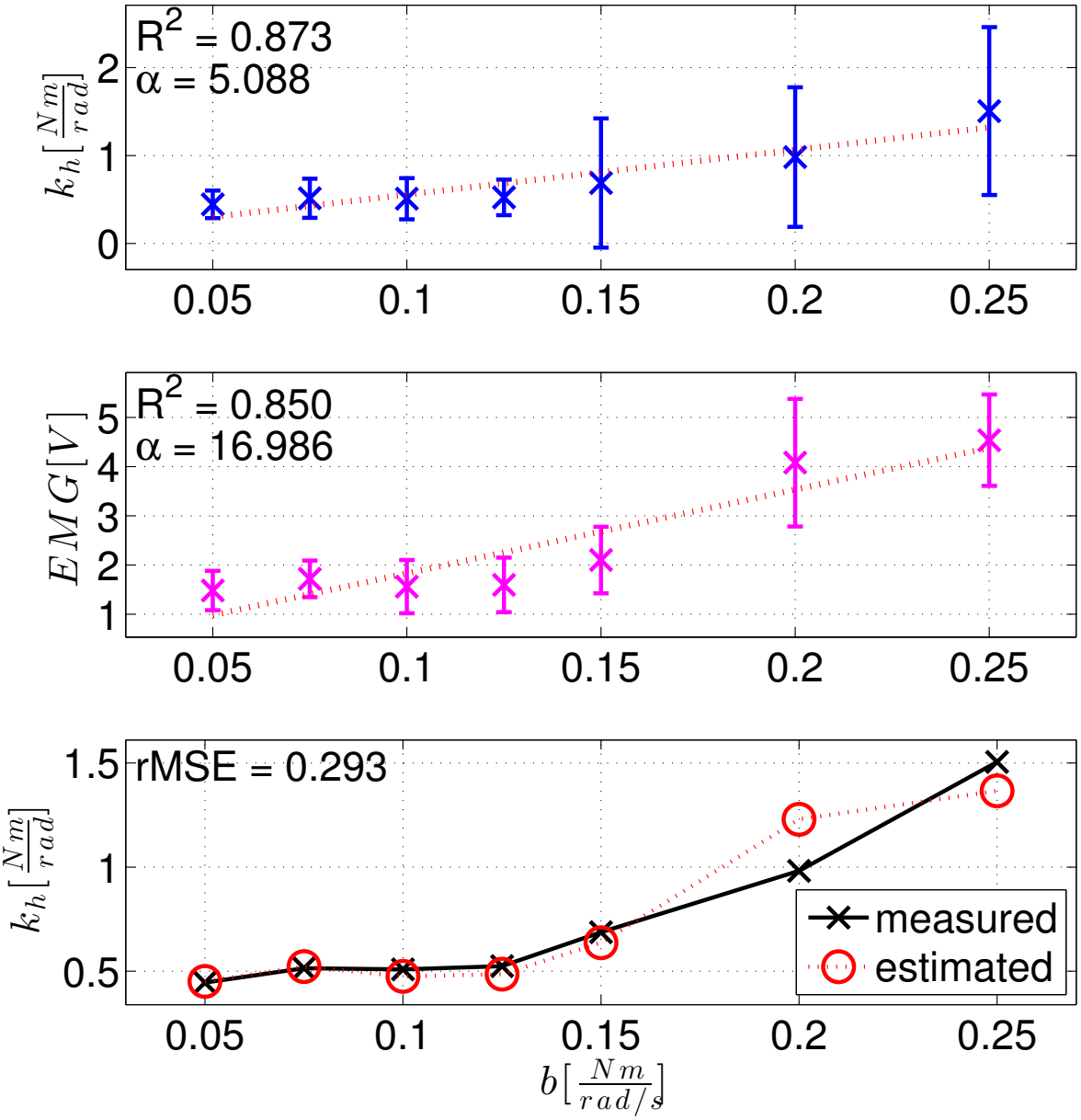

Figure 3.13: Illustrates the experimental results for subject #6. The first two graphs depict the linear fits of the relation between the intensity of the unstable force field and the measured stiffness and muscle activation respectively. The third graph depicts the discrepancy between the perturbation-based stiffness measurements and the sEMG-Based estimations.

4

sEMG-Based Bilateral Control

Now that a method of estimating the operator's limb stiffness from the sEMG values is available (with a previous set of training data) it is time to determine how exactly this estimated stiffness will be incorporated to the Position-Torque architecture seen in "System Architectures", section 2.2.1.

If the architecture for such control scheme is recovered and the sEMG-Stiffness-Estimator incorporated the system looks as the one depicted in Figure 4.1.

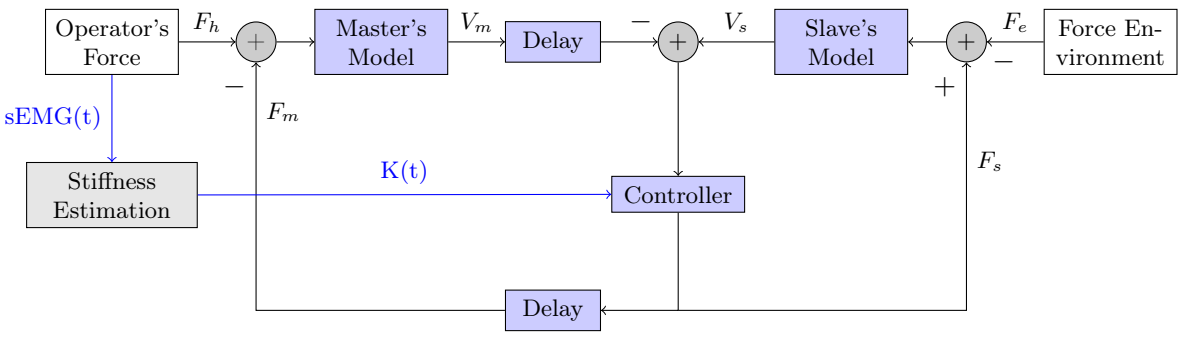

Figure 4.1: Illustrates a bilateral tele manipulation system based on the already discussed Position-Torque architecture where the sEMG-Based stiffness estimation has been incorporated in order to tune the controller in real time according to the operator's estimated limb stiffness.

4.1 Stability

Even though an estimate of the operator's limb is available now it must be kept in mind that what the system is going to tune is the controller's stiffness. Therefore steps must be taken to ensure that whatever the estimated stiffness of the operator's limb is, the system as a whole reminds stable under any case.

It would certainly be of service if the range of valid controller's parameters B and K was known beforehand, a relation that as it has been seen in previous section is drastically constrained by the amount of delay in the transmission.

In order to obtain such values the parameters m and b (mass and damping factor) of the master and slave have to be known. In the case at hand, since the motion is circular instead of a mass an axial inertial moment is present, but no complications arise from this fact, as it has the same effect a mass would on a linear motion. To obtain such parameters a step torque input was applied to the actuators and their movements recorded, $F_{in}(t)$ being the force applied by the motor and $\Phi(t)$ the position of the manipulandum. With the actuators modeled as a mass and a damping factor an analytical expression as a function of m and b for the motion described by the actuator can be reached as it is shown in Equations 4.1.

$$F_{in}(t) = \frac{d^2\Phi(t)}{dt} \cdot m + \frac{d\Phi(t)}{dt} \cdot b \tag{4.1a}$$

$$F_{in}(s) = \mathcal{L}\left\{F_{in}(t)\right\} = \Phi(s)ms^2 + \Phi(s)bs \tag{4.1b}$$

$$\Phi(s) = \frac{F_{in}}{s(ms^2 + bs)} \tag{4.1c}$$

$$\Phi(s) = F_{in} \cdot \left[\frac{1}{bs^2} - \frac{m}{b^2 s} + \frac{m^2}{b^2 ms + b^3}\right] \tag{4.1d}$$

$$\Phi(t) = \mathcal{L}^{-1}\left\{\Phi(s)\right\} \tag{4.1e}$$

$$\Phi(t) = F_{in} \cdot \left[\frac{t}{b} \cdot u(t) - \frac{m}{b^2} \cdot u(t) + \frac{m}{b^2} \cdot e^{-bt/m}\right] \tag{4.1f}$$

Once such analytical response is known an iterative process was designed in order to find out the values of m and b which would make a best fit to the theoretical shape of

the response. This iteration was made using RMSE to evaluate the validity of the curb fitting. The values obtained were:

$$m = 1.189 \quad \left[g \cdot m^2\right] \tag{4.2a}$$

$$b = 0.000568 \quad \left[\frac{Nm}{rad/s}\right] \tag{4.2b}$$

This iterating fitting method can either be done in the Laplace space where the analytical output is nothing more than a polynomial or in the temporal space. The latter one was chosen as the first one, while having some advantages, involved the transformation of a numerical set of data $\Phi(t)$ to the Laplace space, which is not trivial.

With these values, and taking into account the delay of the hardware setup (2ms in total) the maximum values of K and B can be extracted. If this values are used to generate the plots mentioned earlier in Equations 2.19 and 2.20 the graph shown in Figure 4.2 can be obtained.

Despite the fact that according to this results very high values of K are theoretically achievable, once we introduce the parameters in the physical setup, because of the effect of discretitzation, qualification noise, and other effects, values of K higher than twenty turned out to produce small high frequency oscillations, no matter the values of B selected. Because of this reason a value of $B = 0.1$ $[^{Nm}/_{(rad/s)}]$ was used, leading to a margin of stiffness ranging up to the mentioned quantity, twenty.

As mentioned, the margins of stiffness tuning were chosen to be values K raging from 0.1 to 20 $[Nm/rad]$ and a value of B of 0.1 $[^{Nm}/_{(rad/s)}]$ was fixed.

4.2 Implementation

Once the training data has been acquired (the data containing both the perturbation-base stiffness measurements and the sEMG data), sEMG data was dimensionally reduced into a general muscle activation value, the various coefficient computed (α''', β'''), and the operator's stiffness was estimated, this reading has to be incorporated into the controller (as shown in Figure 4.1). However, certain factors have to be taken into account.

First of all, the system is not going to exactly map the operator's stiffness to the controller's stiffness, but a linearization between the two will be performed. This way the controller will be capable of assuming it's maximum and minimum allowed stiffness when the human operator's stiffness reaches it's own maximum and minimum. At this

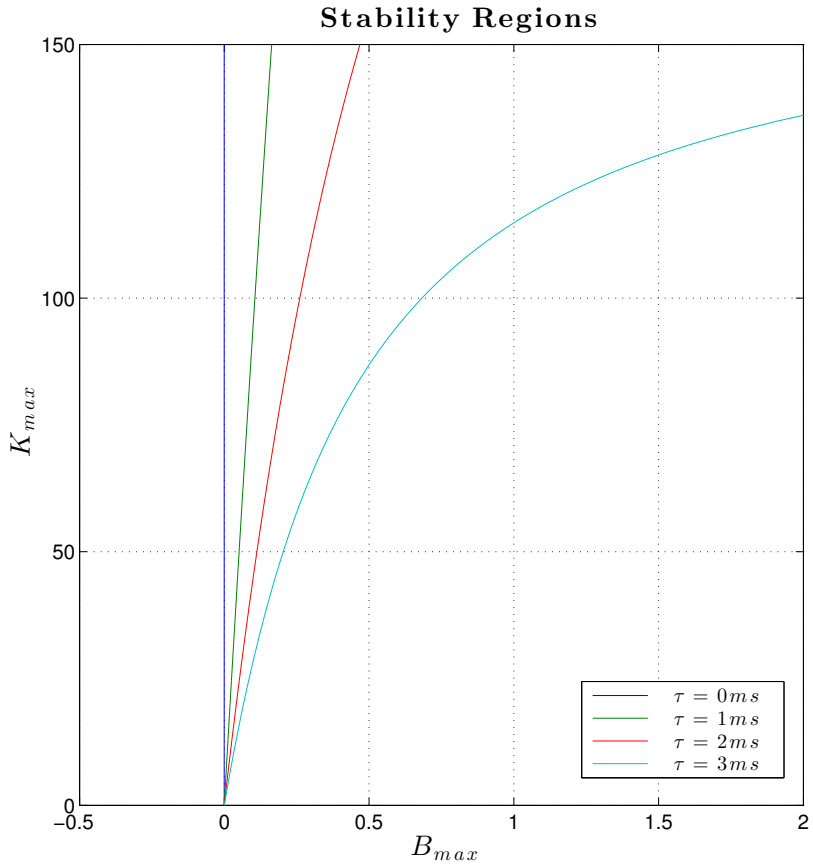

Figure 4.2: This graphs plots the critical values of K in function of B and certain fixed delay, τ for a bilateral system with a delayed transmission where delays are kept small and the parameters obtained from the previous analysis are introduced.

point a problem arises, as even with a set of training data were certain stiffnesses from the operator's limb have been estimated it is not impossible for a higher operator's stiffness to be estimated during normal operation. Even if the operator's stiffness increases beyond a point accounted for in the training it is important (in order to keep the system within the stable region) that the controller's stiffness does not exceed its own maximum. This is accomplished by saturating the controller's stiffness when the result of the linearization between the training set and the controller's stiffness exceeds certain value.

Therefore first the sEMG estimation coefficients (α''', β''') from Equation 3.13 are used to obtain an estimation of the operator's stiffness.

$$\overline{k_h} = \frac{\alpha'}{\alpha''}a + (\beta' - \frac{\alpha'\beta''}{\alpha''}) = \alpha'''a + \beta''' \tag{4.3}$$

Then the estimated stiffness needs to be normalized within the stability boundaries of the system, defined between K_c^{min} and K_c^{max}. This leads to the stiffness controller command as.

$$k_c(t) = (K_c^{max} - K_c^{min}) \cdot \frac{\overline{k_h(t)} - k_h^{min}}{k_h^{max} - k_h^{min}} + K_c^{min}. \tag{4.4}$$

And as stated in the previous section it must be ensured that the commanded controller's stiffness remains within the stability boundaries of the system.

$$K_c^{min} \leq k_c(t) \leq K_c^{max}, \quad \forall t \geq 0 \tag{4.5}$$

4.3 Experimental Validation

In order to validate the whole system a series of final experiments were conducted. First one subject performed the training in order to acquire the principal component or Eigenvector and the linearization factors (α''', β'''), with this his stiffness was estimated as explained in the previous sections.

Once the training data was available the estimated stiffness was linearized and fed to the controller of the system. At that point the subject was told to repeatedly follow a trajectory which ranged from -0.5 rad to 0.5 rad continuously ten times. This exercise was performed both with low-stiffness and high stiffness, in this case the subject was told to either follow the trajectory as if he/she was performing a low precision (low stiffness) task or to do it as a high precission task (high stiffness). Once the results were recorded the position errors between master and slave where evaluated for both cases and the magnitudes of the torques generated during the experiment were compared. The result was as expected for either a low-stiffness controller in the first cases and for a high-stiffness in the second case, validating the system's ability to tune the stiffness of the impedance controller. The results of such test can be seen in Figure 4.3.

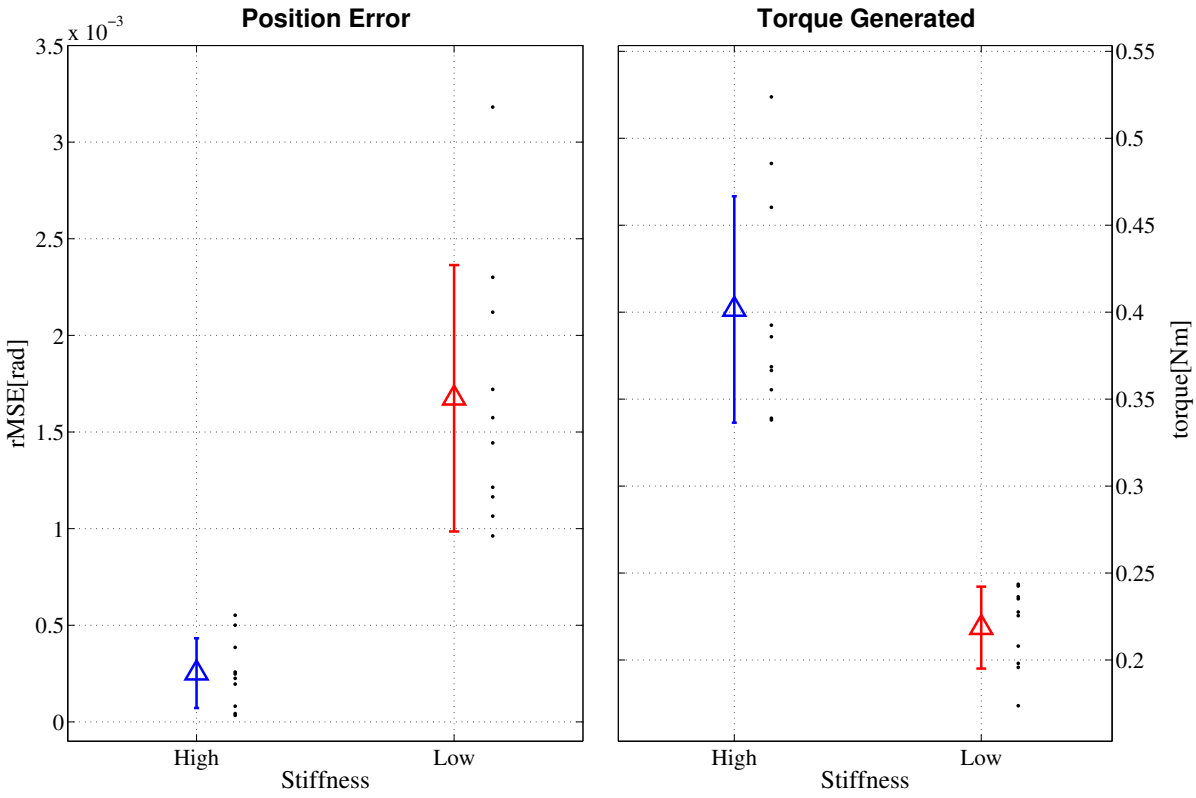

Figure 4.3: This graph plots both the position error (RMSE) and the magnitudes of the torque generated during both exercises, one with a high and one with a low stiffness. In the graph both the means and the standard deviations are plotted

As a further validation another experiment was conducted. After the training the subject was also told to move the master device and reach a certain target position while a very high stiffness object was placed in the trajectory of the slave device. This procedure was also repeated both with a low stiffness and with a high stiffness. The results, which can be seen in Figure 4.4 also shows how upon first contact with the object (which happens under low stiffness) a relatively low torque is generated, while at the time of the second contact, which happened just seconds after under high stiffness a much higher torque is generated to try to compensate the position error. Again these results are consistent with a low-stiffness controller in the first case and with a high-stiffness controller in the second case.

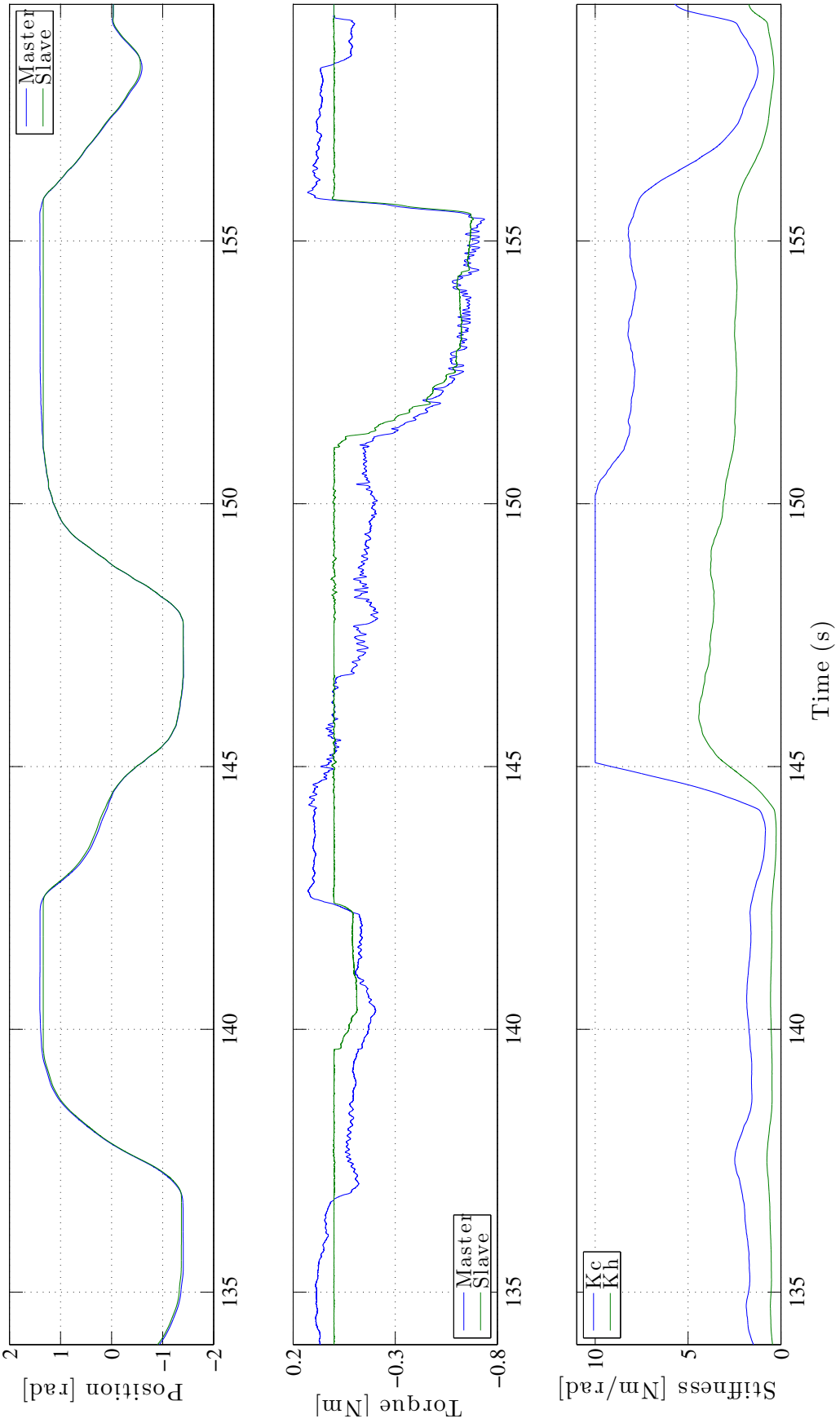

Figure 4.4: This graph plots both the position of the master and slave device, the torques exerted by them, and also the operator's estimated stiffness and the linearization and saturation of such which actually feeds the controller of the system

5

Conclusions

Many conclusions can be extracted from the work presented. For one it is clear that an sEMG-tuned Bilateral Controller is possible to implement and the computational power needed to operate such is relatively small. As stated in the introduction however, the real challenge is evaluating what task can really benefit from such approach.

As future improvements and enhancements to this initial research we could say that the most important problem to be solved would be the capacity of the hardware used. In order to be able to obtain much more precise readings parallel actuators could be used, which are capable of generating greater forces with lower coupled masses and dam pings. Besides the actuators used also different methods other than the presented perturbation-based stiffness measurements could be implemented as such presented in [2]. Also, another enhancement to this work could be to implement some sort of online training, making the system more intelligent and capable of readjusting it's estimation parameters on the go. For example tiny perturbations could be introduced every once in a while (when it is somehow known for that to be safe) and the estimation factors could be recomputed every time a value out of the dynamic range is found.

On the other hand this work could lead to very interesting lines of research. Let's keep in mind that all experiments where based on a 1-DOF device and not taking into account for changes in geometrical stiffness. Imagine however that what has been introduced in this work was extended to take into account the operator's limb position (there are many consumer-electronics sensors capable of detecting human position). Along with the geometrical data a much more precise and general stiffness estimation could be performed, however, the complexity of such system would increase drastically as not only would the

kinematics of the human body would need to be used in order to compute the geometrical stiffness factor at the endpoint but a much bigger set of training data would need to be collected in order to comprehend how each muscle activation factor affects the joint stiffness in a determined geometrical position.

Appendices

A

The Experimental Setup

For all the experiments performed during this work the following experimental setup was used:

- A first PC running SUSE Linux with MatLab and Simulink with the Real-Time Workshop installed.
- A second PC running QNX Neutrino 6.3 Equipped with a double slot CAN bus card and an National Instruments PCI-603E card.
- Two CAN controlled brushless motors equipped with a 1-DOF torque sensor each and a maximum nominal torque of 1.7 Nm.
- Ten MYOBOCK© sEMG electrodes manufactured by OttoBock.

The features of the National Instruments card, as far as this theses is concerned were 16 12-bit analog inputs and a total maximum bitrate of 200KS/s. In the case at hand each electrode was sampled at 500 S/s leading up to an accumulated nitrate of 5KS/s, well below the hardware's maximum. Even though the Analog to Digital Converters (ADC) have a 12-bit output the input of the card is bipolar, while the signal generated by the electrodes is unipolar, this means the final resolution is that of an 11-bit ADC as the whole dynamic range is not exploited.

As mentioned in the previous sections the sEMG electrodes provided a rectified and demodulated version of the real raw sEMG signals with a spectrum ranging from 0 to a few tens of Hertz. In Figure A.1 a simplified scheme of the setup can be seen.

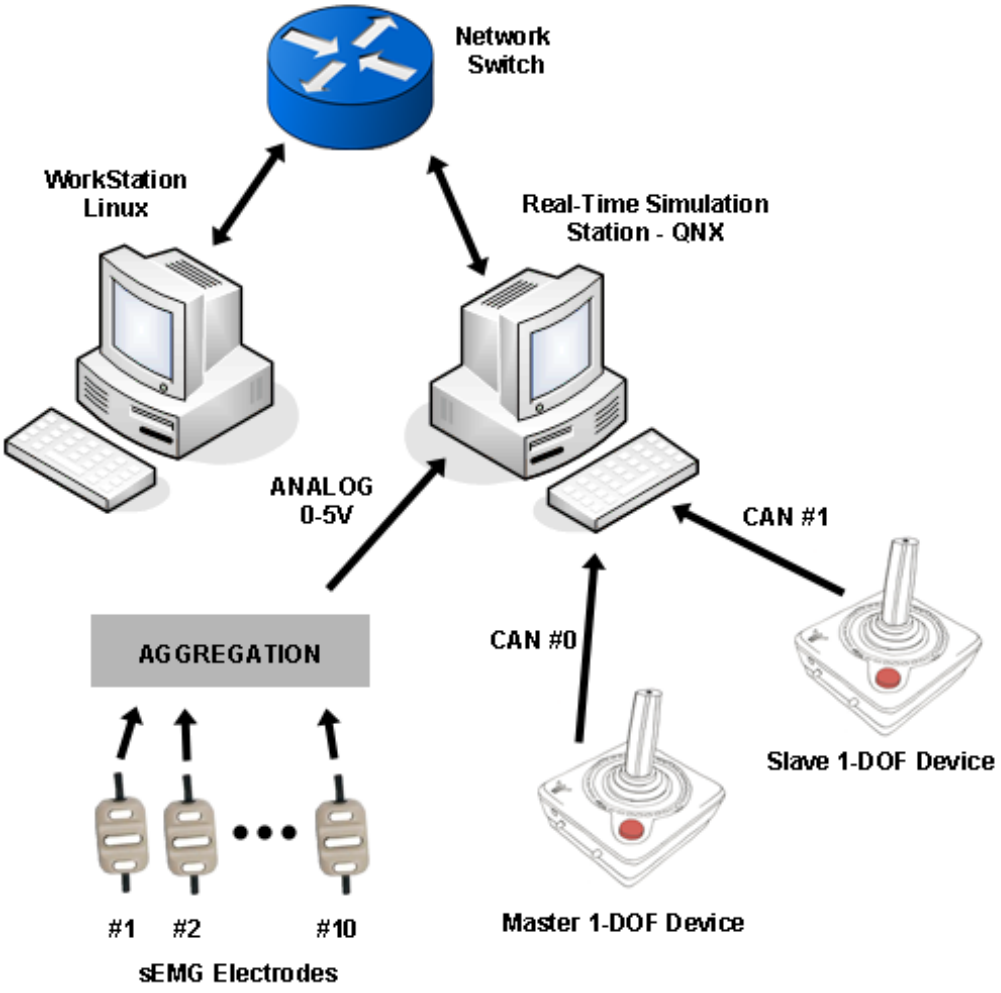

Figure A.1: Here a simplified version of the experimental setup can be seen.

All experiments were programmed using Simulink running on the Linux machine. Once an experiment was to be conducted the model would be compiled for the target platform (QNX) and then executed on the other machine. Simulink's Real-Time Workshop allows for TCP/IP connections between the machine running the model and other machines, by doing this supervision could be carried out on the running model. This utility must not be abused though as it can lead to saturation of the target machine (the one running the model, the QNX machine).

QNX is a Real-Time Operating System (RTOS) which means that it is capable of ensuring the execution of a certain process (in this case the model) every certain period of time, which in the case at hand was one millisecond.

In order for the QNX machine to be able to interface with the hardware (write to the actuators and read from the electrodes and the torque sensors) device drivers and Simulink S-Functions (basically blocks written in C for the case at hand) are required. The drivers and S-Functions for the actuators and torque sensors were provided by previous work buy the drivers and S-Functions for the NI card had to be written along with other S-Functions. This work is described in more detail in annex B and C.

B

Adquisition's Card Device Driver

National Instruments provides device drivers for their acquisition card for Linux and Windows, however, they do not provide any driver for QNX, the OS under which the model runs. Therefore, a PCI device driver had to be written which could retrieve the data from the card and make them available to the OS's processes through a device node.

Unlike Linux and many other Operating Systems, QNX does not require device drivers to run within the kernel space. This is certainly an advantage when it comes to writing a driver as it simplifies the whole process. While NI does not provide a driver for QNX, it does provide a Driver Develeopment Kit (DDK from now on) to help those in the situation at hand to develop drivers for any platform. What NI provides is a series of code which, along with a bus interface middleware let's us execute commands on the board.

The final driver was written in such a way that the board could be configured by writing a filled in structure called "s_config_adq" (which can latter on be seen in the code) to the device node. This structure let's the user configure type of input (differential or single ended), the internal gain of the board, the sampling frequency, the number of samples to be acquired (if a continuous acquisition is not desired), and the number of channels to be sampled. It is important to note that the internal reference clock of the board was used instead of relying on the Operating System's periodic calls, which, even

though running on an RTOS are pretty precise, they always have a certain important jitter when it comes to signal processing purposes.

The driver is invoked by specifying two arguments, first the PCI address where the board resides (an example of such could be "PXI5::14::INSTR" and then a second argument specifying the desired path and name where the device node is to be created, "/dev/ni/board0" for example. At this point, if the attachment to the board is successful, the device driver will switch to background execution and remain still until a configuration command is issued by writing the "s_config_adq" structure to the device node. From that point onwards samples can be read consecutively from the device node in ascending channel order. The PCI-603E card has a sample buffer of 512 positions, if samples are not received at a high enough rate the oldest ones will be lost.

Next the code which had to be written for the device driver is presented. For reference there is also a very verbose help command.

Listing B.1: NiADQ_Data.h

```
#ifndef _DATA_H_
#define _DATA_H_

#define kGNU        1

#define NiADQ_DEBUG    0
#define debug_msg(db, args...) do { if (db) { printf(args); } } while(0)
#define LENGTHBUFFERSAMPLES    10000

typedef struct {
  char  *Device;
  char  *Address;
} s_config_dd;

typedef struct {
  int    Diff;  // 0 = Single ended; 1 = Differential
  int    Gain; //
  int    ScanInterval;
  int    NumberOfSamples;
  int    NumberOfChannels;
  int    Valid;
} s_config_adq;

typedef struct {
  short int Channels[16];
} s_sample;

#endif //_DATA_H_

#endif //_DATA_H_
```

Listing B.2: DD_NiADQ.cpp

```

//#########################################################################
//#########################################################################
```

```
4  //####
5  //####  Developed by Albert Arquer
6  //####
7  //####    DLR - Deutches Zentrum fÃijr Luft-und-Raumfahrt
8  //####    RMC - Robotiks und Mechatroniks Center
9  //####
10 //#########################################################################
11 //####
12 //####  This driver is intended for acquiring data from a Ni
13 //####  PCI6025E card on a QNX  Neutrino 6.3 machine. It
14 //####  is based on the software provided by National
15 //####  Instruments as part of the DDK (Driver Development
16 //####  Kit) and also the examples provided also by Ni.
17 //####
18 //#########################################################################
19
20
21 #include "osiBus.h"
22 #include "tSTC.h"
23 #include "tESeries.h"
24 #include "BoardAPI.h"
25 #include "NiADQ_Data.h"
26
27 #include <stdio.h>
28 #include <time.h>
29 #include <errno.h>
30 #include <stddef.h>
31 #include <stdlib.h>
32 #include <unistd.h>
33 #include <sys/iofunc.h>
34 #include <sys/dispatch.h>
35 #include <string.h>
36 #include <math.h>
37 #include <libc.h>
38
39 int    stoi(char *arg);
40 void   displayhelp();
41 void   *Driver_Thread(void *args);
42 void   Inc_WrPtr();
43 void   Inc_RdPtr();
44 int    DDNiADQ_GetSamples(void);
45 int    DDNiADQ_ReadBoard(resmgr_context_t *ctp,io_read_t *msg,RESMGR_OCB_T *ocb);
46 int    DDNiADQ_WriteBoard(resmgr_context_t *ctp,io_write_t * msg,RESMGR_OCB_T *ocb);
47 int    DDNiADQ_OpenBoard(resmgr_context_t *ctp,io_open_t *msg,RESMGR_HANDLE_T *handle
       ,void *extra);
48
49 static resmgr_connect_funcs_t      connect_funcs_board;
50 static resmgr_io_funcs_t           io_funcs_board;
51 static iofunc_attr_t                 attr_board;
52 static pthread_attr_t        Thread_Attr;
53 int        wr_ch_num = 0, read_flag = 1, first_read_after_config = 0;
54 s_config_dd    ConfigDD;
55 s_config_adq     ConfigADQ;
56 s_sample       Sample;
57 tESeries       *board;
58 tSTC       *theSTC;
59
```

```
60
61
62  int main(int argc, char **argv) {
63
64    tAddressSpace      bar1;
65      static iBus*     bus;
66      pthread_t        t2;
67
68
69    if (argc < 2 || argc > 3) {
70      printf("\n\r\t[ERROR] Incorrect number of arguments! %d", argc);
71      displayhelp();
72      printf("\n\n\r");
73      return NULL;
74    }
75
76    ConfigDD.Address = argv[1]; //PCI_Address PXI5::4::INSTR
77    ConfigDD.Device = argv[2];  // /dev/ni/board0
78
79    debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] Starting NiADQ Device Driver...");
80    debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] Using device %s and device node %s",
        ConfigDD.Address, ConfigDD.Device);
81
82      bus = acquireBoard(ConfigDD.Address);
83    if(bus == NULL) {
84        printf("\n\r\t[ERROR] Error accessing the PCI device %s. Exiting.\n\n\r",
        ConfigDD.Address);
85        return NULL;
86      }
87
88    ThreadCtl (_NTO_TCTL_IO,NULL);
89
90    //Intitialise Mite Chip.
91    BoardAPI_InitMite(bus);
92
93    bar1 = bus->createAddressSpace(kPCI_BAR1);
94    board = new tESeries(bar1);
95    theSTC = new tSTC(bar1);
96
97    pthread_create(&t2, NULL, Driver_Thread, NULL);
98    pthread_join(t2,NULL);
99
100   delete theSTC;
101   delete board;
102   bus->destroyAddressSpace(bar1);
103
104   return 0;
105 }
106
107
108
109 void displayhelp(void) {
110
111 printf("\n\r\t");
112 printf("\n\r\t=============================================================");
113 printf("\n\r\t                          HELP                               ");
114 printf("\n\r\t=============================================================");
```

```
115
116 printf("\n\r\t");
117 printf("\n\r\tUsage: pci_address device_node\n\r");
118 printf("\n\r\t  >pci_address:");
119 printf(" \r\t\t\t\t Device address, (example PXI5::14::INSTR)");
120 printf("\n\r\t  >device_node:");
121 printf("\r\t\t\t\t Path and the filename of the device node to be");
122 printf("\n\r\t\t\t\t created for the device. For example, /dev/ni/board0 would");
123 printf("\n\r\t\t\t\t create a  node named \"board0\" in the /dev/ni directory.");
124 printf("\n\r");
125 printf("\n\r\tThe execution of the driver will attach to the pci device and");
126 printf("create the device node, however no data will be sampled until");
127 printf("\n\r\t a command is written to the device node. The command");
128 printf("consists of a filled-in \n\r\t\"s_config_adq\" structure");
129 printf("(please see NiADQ_Data.h file). The structures fields are:");
130 printf("\n\r");
131 printf("\n\r\t  >Diff:");
132 printf("\r\t\t\t\t This parameter indicates adquisition in ingle-ended or diff");
133 printf("\n\r\t\t\t\t Use 0 for single ended and 1 for differential. \n\r\t\t\t If");
134 printf("1 is used differential CH0 will be read as Channel0 - Channel1,");
135 printf("\n\r\t\t\t\tdifferential CH1 as Channel2 - Channel3 and so on.");
136 printf("\n\r\t  >gain:");
137 printf("\r\t\t\t\t This parameter sets the gain to be applied. The \n\r\t\t\t\t");
138 printf("allowed values go from 1 to 4 and imply a gain of 0.5, 1, 10 ");
139 printf("and 100 \n\r\t\t\t\t respectively.");
140 printf("\n\r\t  >scan_interval:");
141 printfd("\r\t\t\t\t This parameter sets the interval in 20MHz tics that should");
142 printf("\n\r\t\t\t\t exist between each sample. The minimum is 2000 tics ");
143 printf("\n\r\t\t\t\t and the maximum 200000. This leads to sampling ");
144 printf("frecuencies between 10KSps and 100Sps");
145 printf("\n\r\t  >NumberOfSamples: \r\t\t\t\tNumber of samples that will be");
146 printf("acquired from each channel. Enter 0 for continuous acquisition");
147 printf("\n\r\t  >NumberOfChannels:\r\t\t\t\tNumber of channels ");
148 printf("(starting from Channel0 upwards) that will be acquired.");
149
150 printf("\n\r\t");
151 printf("\n\r\t eg. ./DD_NiADQ PXI5::14::INSTR /dev/ni/board0");
152
153 printf("\n\r\t");
154 printf("\n\r\tFollowing there is a list of all the Ni PCI cards found:\n");
155 BoardAPI_ListPCI();
156
157 }
158
159
160
161 void *Driver_Thread(void *args) {
162
163   resmgr_attr_t          resmgr_attr_board;
164   dispatch_t             *dpp;
165   dispatch_context_t     *ctp;
166   int                    id_board;
167
168
169   ThreadCtl (_NTO_TCTL_IO,NULL);
170
171   debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] Starting Board Ressource Manager.");
```

```
172
173   if((dpp = dispatch_create()) == NULL) {
174       fprintf(stderr, "\n\r\t[ERROR] Unable to allocate dispatch handle.");
175     return NULL;
176   }
177
178   memset(&resmgr_attr_board, 0, sizeof resmgr_attr_board);
179   resmgr_attr_board.nparts_max = 1;
180   resmgr_attr_board.msg_max_size = 2048;
181
182   iofunc_func_init(_RESMGR_CONNECT_NFUNCS, &connect_funcs_board,_RESMGR_IO_NFUNCS, &
      io_funcs_board);
183   connect_funcs_board.open = DDNiADQ_OpenBoard;
184   io_funcs_board.read = DDNiADQ_ReadBoard;
185   io_funcs_board.write = DDNiADQ_WriteBoard;
186
187   iofunc_attr_init(&attr_board, S_IFNAM | 0666, 0, 0);
188
189   if((id_board = resmgr_attach(dpp, &resmgr_attr_board, ConfigDD.Device,_FTYPE_ANY,
      0,&connect_funcs_board, &io_funcs_board, &attr_board)) == -1) {
190     fprintf(stderr, "\n\r\t[ERROR] Unable to attach BOARD name, maybe it is already
      _open.\n");
191       return NULL;
192   }
193
194   ctp = dispatch_context_alloc(dpp);
195   ConfigADQ.Valid = 0;
196
197   debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] Board Ressource Manager up and running");
198
199   while(1) {
200     if((ctp = dispatch_block(ctp)) == NULL) {
201       fprintf(stderr, "\n\r\t[ERROR] Block error, exiting");
202         return NULL;
203     }
204     dispatch_handler(ctp);
205     DDNiADQ_GetSamples();
206     }
207 }
208
209
210
211 int DDNiADQ_GetSamples() {
212
213   while ((!(theSTC->AI_Status_1.readRegister() & 0x1000)) && (ConfigADQ.Valid == 1))
       {
214
215     Sample.Channels[wr_ch_num] = (short int)board->AIFifoData.readRegister();
216     wr_ch_num++;
217     if (wr_ch_num == ConfigADQ.NumberOfChannels) { wr_ch_num = 0; }
218     read_flag = 0;
219     }
220
221     return 0;
222 }
223
224
```

```
225
226 int DDNiADQ_WriteBoard(resmgr_context_t *ctp, io_write_t *msg,RESMGR_OCB_T *ocb) {
227
228   int      i, CommandsRead, BytesRead, status;
229
230   debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] Data written to the driver...");
231
232     if ((status = iofunc_write_verify(ctp, msg, ocb, NULL)) != EOK)    return (status
      );
233     if ((msg->i.xtype & _IO_XTYPE_MASK) != _IO_XTYPE_NONE)          return(ENOSYS);
234
235   if (msg->i.nbytes >= sizeof(s_config_adq)) {
236
237     CommandsRead = floorf((float)(msg->i.nbytes)/(sizeof(s_config_adq)));
238     BytesRead = CommandsRead*sizeof(s_config_adq);
239
240       _IO_SET_WRITE_NBYTES (ctp, BytesRead);
241
242     for ( i = 0 ; i < CommandsRead ; i++ ) {
243           resmgr_msgread(ctp, &ConfigADQ, sizeof(s_config_adq), sizeof(msg->i));
244     }
245
246       debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] Number of commands read: %d",
      CommandsRead);
247       debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] ConfigDD.Diff = %d", ConfigADQ.Diff);
248       debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] ConfigDD.Gain = %d", ConfigADQ.Gain);
249       debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] ConfigDD.ScanInterval = %d", ConfigADQ.
      ScanInterval);
250       debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] ConfigDD.NumberOfSamples = %d",
      ConfigADQ.NumberOfSamples);
251       debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] ConfigDD.NumberOfChannels = %d",
      ConfigADQ.NumberOfChannels);
252
253     if( (ConfigADQ.Diff < 0) || (ConfigADQ.Diff > 1) || (ConfigADQ.Gain < 1) || (
      ConfigADQ.Gain > 4) ||
254       (ConfigADQ.ScanInterval < 2000) || (ConfigADQ.ScanInterval > 200000) || (
      ConfigADQ.NumberOfSamples < 0) || (ConfigADQ.NumberOfChannels < 1) ||
255       ((ConfigADQ.Diff == 1) && (ConfigADQ.NumberOfChannels > 8)) || ((ConfigADQ.
      Diff == 0) && (ConfigADQ.NumberOfChannels > 16))) {
256       printf("\n\r\t[ERROR] Invalid arguments!");
257       displayhelp();
258       printf("\n\n\r");
259       return NULL;
260     }
261
262     ConfigADQ.Valid = 0;
263     wr_ch_num = 0;
264     read_flag = 0;
265     first_read_after_config = 0;
266
267     BoardAPI_ConfigureBoard(theSTC, board, &ConfigADQ);
268     BoardAPI_MSCClockConfigure(theSTC);
269     BoardAPI_ClearFIFO(theSTC);
270     BoardAPI_ResetAll(theSTC);
271     BoardAPI_BoardPersonalize(theSTC);
272     BoardAPI_InitializeConfigurationMemoryOutput(theSTC);
273     BoardAPI_TriggerSignals(theSTC, &ConfigADQ);
```

```
274       BoardAPI_NumberOfScans(theSTC, &ConfigADQ);
275       BoardAPI_ScanStart(theSTC, &ConfigADQ);
276       BoardAPI_EndOfScan(theSTC, &ConfigADQ);
277       BoardAPI_ConvertSignal(theSTC);
278       BoardAPI_ClearFIFO(theSTC);
279       BoardAPI_Arming(theSTC);
280
281       ConfigADQ.Valid = 1;
282
283       debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] The card was reset and the adquisition re-
          started...");
284
285     } else {
286       debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] Too few bytes written to the driver...
          Data discarded.");
287     }
288
289
290     if (msg->i.nbytes > 0)  ocb->attr->flags |= IOFUNC_ATTR_MTIME | IOFUNC_ATTR_CTIME;
291     return (_RESMGR_NPARTS (0));
292 }
293
294
295
296 int DDNiADQ_ReadBoard(resmgr_context_t *ctp, io_read_t *msg,RESMGR_OCB_T *ocb) {
297
298     int nbytes;
299     int status;
300
301     if ((status = iofunc_read_verify (ctp, msg, ocb, NULL)) != EOK)    return (status);
302     if ((msg->i.xtype & _IO_XTYPE_MASK) != _IO_XTYPE_NONE)          return (ENOSYS);
303
304     if (first_read_after_config == 0) {
305
306       memset(&Sample, NULL, sizeof(s_sample));
307
308       nbytes  = sizeof(s_sample);
309         SETIOV (ctp->iov, (void *) &Sample, sizeof(s_sample));
310         _IO_SET_READ_NBYTES (ctp, nbytes);
311       first_read_after_config = 1;
312
313       BoardAPI_StartTheAcquisition(theSTC);
314
315       return (_RESMGR_NPARTS (1));
316
317     } else {
318
319       if (wr_ch_num == 0 && read_flag == 0) {
320
321           nbytes  = sizeof(s_sample);
322           SETIOV (ctp->iov, (void *) &Sample, sizeof(s_sample));
323           _IO_SET_READ_NBYTES (ctp, nbytes);
324
325         read_flag = 1;
326
327           return (_RESMGR_NPARTS (1));
328
```

```
329     } else {
330
331         return (_RESMGR_NPARTS (0));
332     }
333   }
334 }
335
336
337
338 int DDNiADQ_OpenBoard (resmgr_context_t *ctp, io_open_t *msg, RESMGR_HANDLE_T *
      handle, void *extra) {
339
340     if (handle->count > 0) {
341         printf("\n\r\t[ERROR] Ressource already opened somewhere.");
342           return (EBUSY);
343     }
344
345     return (iofunc_open_default (ctp, msg, handle, extra));
346 }
347
348
349
350 int  stoi(char *arg) {
351
352   int i = 0, ret = 0;
353
354   while (arg[i] != '\0') {
355     if (i != 0 || arg[i] != '-') {
356       ret = ret*10 + (arg[i] - '0');
357     }
358     i++;
359   }
360
361   if (arg[0] == '-') {
362     ret = -ret;
363   }
364
365   return ret;
366 }
```

Listing B.3: BoardAPI.h

```
1  #ifndef _BOARDAPI_H_
2  #define _BOARDAPI_H_
3
4
5  #include <stdio.h>
6  #include "osiBus.h"
7  #include "tESeries.h"
8  #include "tSTC.h"
9  #include "NiADQ_Data.h"
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <string.h>
14
15 #include <hw/pci.h>
```

```
#include <sys/types.h>

const uint16_t kPCIVendorIdNationalInstruments = 0x1093;
const uint16_t kPCIDeviceId_6250 = 0x70b4;
const uint16_t kPCIDeviceId_6602 = 0x1310;


int    BoardAPI_ListPCI(void);
void   BoardAPI_InitMite(iBus *bus);                          //Initialise Mite Chip.
void   BoardAPI_ConfigureBoard(tSTC *theSTC, tESeries *board, s_config_adq *ConfigADQ
    );
void   BoardAPI_MSCClockConfigure(tSTC *theSTC);
void   BoardAPI_ResetAll(tSTC *theSTC);
void   BoardAPI_BoardPersonalize(tSTC *theSTC);
void   BoardAPI_InitializeConfigurationMemoryOutput(tSTC *theSTC);
void   BoardAPI_BoardEnvironmentalize(tSTC *theSTC);
void   BoardAPI_TriggerSignals(tSTC *theSTC, s_config_adq *ConfigADQ);
void   BoardAPI_ScanStart(tSTC *theSTC, s_config_adq *ConfigADQ);
void   BoardAPI_EndOfScan(tSTC *theSTC, s_config_adq *ConfigADQ);
void   BoardAPI_ClearFIFO(tSTC *theSTC);
void   BoardAPI_StartTheAcquisition(tSTC *theSTC);
void   BoardAPI_NumberOfScans(tSTC *theSTC, s_config_adq *ConfigADQ);
void   BoardAPI_ConvertSignal(tSTC *theSTC);
void   BoardAPI_Arming(tSTC *theSTC);

#endif //_BOARDAPI_H_
```

Listing B.4: BoardAPI.cpp

```
#include <stdio.h>
#include <errno.h>

#include "BoardAPI.h"
#include "NiADQ_Data.h"
#include "tESeries.h"
#include "tSTC.h"
#include "osiBus.h"


extern int errno;



void BoardAPI_InitMite(iBus *bus) {

  tAddressSpace  bar0;
  u32 physicalBar1;

  debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] Initialization Mite Chip.");

  if(!bus->get(kIsPciPxiBus,0)) return;                                    //Skip
    MITE initialization for PCMCIA boards
                                                          //(which do not have a MITE
    DMA controller)
  bar0 = bus->createAddressSpace(kPCI_BAR0);

```

```
26    physicalBar1 = bus->get(kBusAddressPhysical,kPCI_BAR1);
        //Get the physical address of the DAQ board
27    bar0.write32(0xC0, (physicalBar1 & 0xffffff00L) | 0x80);
        //Tell the MITE to enable BAR1, where the rest of the board's registers are
28
29    bus->destroyAddressSpace(bar0);
30
31    debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] Mite Chip Initialized.");
32  }
33
34
35
36  int BoardAPI_ListPCI(void) {
37
38      debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] List devices for QNX Neutrino:");
39
40    int    pidx;
41      void*    hdl;
42      int    phdl;
43      struct pci_dev_info info;
44
45      phdl = pci_attach( 0 );                                    /*
       ------------------------Connect to the PCI server ---------------------*/
46      if( phdl == -1 ) {
47          fprintf( stderr, "\n\r\t[ERROR] Unable to connect to the PCI server." );
48          return EXIT_FAILURE;
49      }
50      debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] Now connected to the PCI server.");
51
52
53      memset( &info, 0, sizeof( info ) );                          /*
       ------------ Initialize the pci_dev_info structure ------------------*/
54      pidx = 0;
55      info.VendorId = kPCIVendorIdNationalInstruments;
56      info.DeviceId = kPCIDeviceId_6250;
57
58    debug_msg(NiADQ_DEBUG, "\n\r\t[DEBUG] Looking for any Ni cards...");
59    hdl = (void*) 1;
60    while(hdl != 0){
61        hdl = pci_attach_device( NULL, PCI_INIT_ALL | PCI_SHARE | PCI_SEARCH_VEND ,
      pidx, &info );
62        pidx++;
63        if( hdl == NULL && pidx == 1 ) {
64            fprintf( stderr, "\n\r\t[ERROR] Unable to locate any boards from Ni...[%d]
      ", errno);
65            if (errno == EBUSY)    printf("Device Busy.");
66            if (errno == EINVAL)   printf("Could not attach.");
67            if (errno == ENODEV)   printf("Device not found.");
68        } else if ( hdl != NULL ) {
69          printf ("\n\r\t > Board Nr.#%d: PXI%d::%d::INSTR\tDevID: 0x%04X\tIRQ: 0x%04X
      ",
70            pidx,
71                info.BusNumber,
72                PCI_DEVNO(info.DevFunc),
73                info.DeviceId,info.Irq);
74
75            pci_detach_device(hdl);
```

```
76            ++pidx;
77          }
78    }
79
80
81      pci_detach( phdl );                                          /*
      ------------------------ Disconnect to the PCI server ------------------*/
82
83      return EXIT_SUCCESS;
84  }
85
86
87
88  void BoardAPI_ConfigureBoard(tSTC *theSTC, tESeries *board, s_config_adq *ConfigADQ)
        {
89
90    int i;
91
92    //Clear configuration memory
93    theSTC->Write_Strobe_0.writeRegister(0x0001);
94
95    //Clear ADC FIFO
96    theSTC->Write_Strobe_1.writeRegister(0x0001);
97
98    for ( i = 0 ; i < ConfigADQ->NumberOfChannels ; i++ ) {
99
100     board->ConfigFifoHigh.setChannel(i);
101     board->ConfigFifoHigh.setBank(0);
102     if (ConfigADQ->Diff == 1) {
103       board->ConfigFifoHigh.setChannelType(board->ConfigFifoHigh.
      kChannelTypeDifferential);
104     } else {
105       board->ConfigFifoHigh.setChannelType(board->ConfigFifoHigh.kChannelTypeRSE);
106     }
107     board->ConfigFifoHigh.flush();
108
109     //Writing to Config_Memory_Low_Register for following channel 0 settings
110     if (i == ConfigADQ->NumberOfChannels - 1) {
111       board->ConfigFifoLow.setLastChannel(1);
112     } else {
113       board->ConfigFifoLow.setLastChannel(0);
114     }
115     board->ConfigFifoLow.setGeneralTrigger(0);                                    //This
       might be interesting.....
116     switch (ConfigADQ->Gain) {
117       case 1: board->ConfigFifoLow.setGain(board->ConfigFifoLow.kGain000_5); break;
118       case 2: board->ConfigFifoLow.setGain(board->ConfigFifoLow.kGain001_0); break;
119       case 3: board->ConfigFifoLow.setGain(board->ConfigFifoLow.kGain010_0); break;
120       case 4: board->ConfigFifoLow.setGain(board->ConfigFifoLow.kGain100_0); break;
121       default: board->ConfigFifoLow.setGain(board->ConfigFifoLow.kGain001_0); break;
122     }
123     board->ConfigFifoLow.setPolarity(board->ConfigFifoLow.kPolarityBipolar);
              //Apparently the E series boards do not support unipolar inputs
124     board->ConfigFifoLow.setDither(0);
125     board->ConfigFifoLow.flush();
126   }
127
```

```
128
129   return;
130 }
131
132
133
134 void BoardAPI_MSCClockConfigure(tSTC *theSTC) {
135
136   theSTC->Clock_and_FOUT.setSlow_Internal_Timebase(1);                    //
        This bit enables the slow internal clock IN_TIMEBASE2 = IN_TIMEBASE/100
137   theSTC->Clock_and_FOUT.setSlow_Internal_Time_Divide_By_2(1);
        //This bit enables the divistion of IN_TIMEBASE2 by 2
138   theSTC->Clock_and_FOUT.setClock_To_Board(1);                            //This
        bit enables the IN_TIMEBASE to feedback or feedthrough to the board through the
        OUTBRD_OSC pin.
139   theSTC->Clock_and_FOUT.setClock_To_Board_Divide_By_2(1);
        //This bit enables the IN_TIMEBASE to be divided by 2 when it is feeded-through.
140   theSTC->Clock_and_FOUT.flush();
141   return;
142 }
143
144
145
146 void BoardAPI_ClearFIFO(tSTC *theSTC) {
147
148   theSTC->Write_Strobe_1.writeRegister(0x0001);
149   return;
150 }
151
152
153
154 void BoardAPI_ResetAll(tSTC *theSTC) {
155
156   theSTC->Joint_Reset.setAI_Reset(1);//Reset important registers
157   theSTC->Joint_Reset.setAI_Configuration_Start(1);//Starting AI configuration
158   theSTC->Joint_Reset.flush();
159
160   //Acknowledges the following interrupt request if the interrupt enable is set
161   theSTC->Interrupt_A_Ack.setAI_SC_TC_Error_Confirm(1);
162   theSTC->Interrupt_A_Ack.setAI_SC_TC_Interrupt_Ack(1);
163   theSTC->Interrupt_A_Ack.setAI_START1_Interrupt_Ack(1);
164   theSTC->Interrupt_A_Ack.setAI_START2_Interrupt_Ack(1);
165   theSTC->Interrupt_A_Ack.setAI_START_Interrupt_Ack(1);
166   theSTC->Interrupt_A_Ack.setAI_STOP_Interrupt_Ack(1);
167   theSTC->Interrupt_A_Ack.setAI_Error_Interrupt_Ack(1);
168   theSTC->Interrupt_A_Ack.flush();
169
170   //Enables Start or Stop Analog Input Operation
171   theSTC->AI_Mode_1.setReserved_One(1);
172   theSTC->AI_Mode_1.setAI_Start_Stop(1);
173   theSTC->AI_Mode_1.flush();
174
175   //Ending AI configuration
176   theSTC->Joint_Reset.setAI_Configuration_Start(0);
177   theSTC->Joint_Reset.setAI_Configuration_End(1);
178   theSTC->Joint_Reset.flush();
179   return;
```

```
180 }
181
182
183
184 void BoardAPI_BoardPersonalize(tSTC *theSTC) {
185
186   theSTC->Joint_Reset.writeAI_Configuration_Start(1);
187
188   theSTC->Clock_and_FOUT.setAI_Source_Divide_By_2(0);
189   theSTC->Clock_and_FOUT.setAI_Output_Divide_By_2(1);
190   theSTC->Clock_and_FOUT.flush();
191
192   theSTC->AI_Personal.setAI_CONVERT_Pulse_Timebase(theSTC->AI_Personal.
        kAI_CONVERT_Pulse_TimebasePulse_Width);
193   theSTC->AI_Personal.setAI_CONVERT_Pulse_Width(theSTC->AI_Personal.
        kAI_CONVERT_Pulse_WidthAbout_1_Clock_Period);
194   theSTC->AI_Personal.setAI_FIFO_Flags_Polarity(theSTC->AI_Personal.
        kAI_FIFO_Flags_PolarityActive_Low );
195   theSTC->AI_Personal.setAI_LOCALMUX_CLK_Pulse_Width(theSTC->AI_Personal.
        kAI_LOCALMUX_CLK_Pulse_WidthAbout_1_Clock_Period);
196   theSTC->AI_Personal.setAI_AIFREQ_Polarity(theSTC->AI_Personal.
        kAI_AIFREQ_PolarityActive_High);
197   theSTC->AI_Personal.setAI_SHIFTIN_Polarity(theSTC->AI_Personal.
        kAI_SHIFTIN_PolarityActive_Low);
198   theSTC->AI_Personal.setAI_SHIFTIN_Pulse_Width(theSTC->AI_Personal.
        kAI_SHIFTIN_Pulse_WidthAbout_2_Clock_Periods );
199   theSTC->AI_Personal.setAI_EOC_Polarity(theSTC->AI_Personal.
        kAI_EOC_PolarityRising_Edge);
200   theSTC->AI_Personal.setAI_SOC_Polarity(theSTC->AI_Personal.
        kAI_SOC_PolarityFalling_Edge);
201   theSTC->AI_Personal.setAI_Overrun_Mode(theSTC->AI_Personal.
        kAI_Overrun_ModeSOC_To_SHIFTIN_Trailing_Edge);
202   theSTC->AI_Personal.flush();
203
204   theSTC->AI_Output_Control.setAI_CONVERT_Output_Select(theSTC->AI_Output_Control.
        kAI_CONVERT_Output_SelectActive_Low);
205   theSTC->AI_Output_Control.setAI_SC_TC_Output_Select(theSTC->AI_Output_Control.
        kAI_SC_TC_Output_SelectActive_High );
206   theSTC->AI_Output_Control.setAI_SCAN_IN_PROG_Output_Select(theSTC->
        AI_Output_Control.kAI_SCAN_IN_PROG_Output_SelectActive_High);
207   theSTC->AI_Output_Control.setAI_LOCALMUX_CLK_Output_Select(theSTC->
        AI_Output_Control.kAI_LOCALMUX_CLK_Output_SelectActive_Low);
208   theSTC->AI_Output_Control.flush();
209
210   theSTC->Joint_Reset.setAI_Configuration_Start(0);
211   theSTC->Joint_Reset.setAI_Configuration_End(1);
212   theSTC->Joint_Reset.flush();
213   return;
214 }
215
216
217
218 void BoardAPI_InitializeConfigurationMemoryOutput(tSTC *theSTC) {
219
220   theSTC->AI_Command_1.writeAI_CONVERT_Pulse(1);
221   return;
222 }
```

```
223
224
225 //I don't think this is needed -> future delet
226 //void BoardAPI_BoardEnvironmentalize(tSTC *theSTC) {
227
228 //  theSTC->Joint_Reset.writeAI_Configuration_Start(1);
229
230 //  theSTC->AI_Mode_2.writeAI_External_MUX_Present(theSTC->AI_Mode_2.
    kAI_External_MUX_PresentEvery_Convert);
231
232 //  theSTC->Joint_Reset.setAI_Configuration_Start(0);
233 //  theSTC->Joint_Reset.setAI_Configuration_End(1);
234 //  theSTC->Joint_Reset.flush();
235 //  return;
236 //}
237
238
239
240 void BoardAPI_TriggerSignals(tSTC *theSTC, s_config_adq *ConfigADQ) {
241
242   theSTC->Joint_Reset.writeAI_Configuration_Start(1);
243
244   //Controls the retriggerability of Counters
245   if (ConfigADQ->NumberOfSamples == 0) {
246     theSTC->AI_Mode_1.writeAI_Trigger_Once(0);
247     theSTC->AI_Mode_1.writeAI_Continuous(1);
248   } else {
249     theSTC->AI_Mode_1.writeAI_Trigger_Once(1);
250     theSTC->AI_Mode_1.writeAI_Continuous(0);
251   }
252
253   //Selects and configures the functanality of START1 trigger
254   theSTC->AI_Trigger_Select.setAI_START1_Select(theSTC->AI_Trigger_Select.
      kAI_START1_SelectPulse);
255   theSTC->AI_Trigger_Select.setAI_START1_Polarity(theSTC->AI_Trigger_Select.
      kAI_START1_PolarityActive_High_Or_Rising_Edge );
256   theSTC->AI_Trigger_Select.setAI_START1_Edge(1);
257   theSTC->AI_Trigger_Select.setAI_START1_Sync(1);
258   theSTC->AI_Trigger_Select.flush();
259
260   theSTC->Joint_Reset.setAI_Configuration_Start(0);
261   theSTC->Joint_Reset.setAI_Configuration_End(1);
262   theSTC->Joint_Reset.flush();
263
264   return;
265 }
266
267
268
269 void BoardAPI_NumberOfScans(tSTC *theSTC, s_config_adq *ConfigADQ) {
270
271   if (ConfigADQ->NumberOfSamples != 0) {
272     theSTC->Joint_Reset.writeAI_Configuration_Start(1);
273
274     theSTC->AI_Mode_2.setAI_SC_Initial_Load_Source(theSTC->AI_Mode_2.
      kAI_SC_Initial_Load_SourceLoad_A);
```

```
275     theSTC->AI_Mode_2.setAI_SC_Reload_Mode(theSTC->AI_Mode_2.
        kAI_SC_Reload_ModeNo_Change);
276     theSTC->AI_SC_Load_A.writeRegister(ConfigADQ->NumberOfSamples);
277     theSTC->AI_Command_1.writeAI_SC_Load(1);
278
279     theSTC->Joint_Reset.setAI_Configuration_Start(0);
280     theSTC->Joint_Reset.setAI_Configuration_End(1);
281     theSTC->Joint_Reset.flush();
282   }
283
284   return;
285 }
286
287
288
289 void BoardAPI_ScanStart(tSTC *theSTC, s_config_adq *ConfigADQ) {
290
291   theSTC->Joint_Reset.writeAI_Configuration_Start(1);
292
293   //Setting the bitfields corresponding to START triggers
294   theSTC->AI_START_STOP_Select.setAI_START_Select(theSTC->AI_START_STOP_Select.
        kAI_START_SelectSI_TC);
295   theSTC->AI_START_STOP_Select.setAI_START_Edge(1);
296   theSTC->AI_START_STOP_Select.setAI_START_Sync(1);
297   theSTC->AI_START_STOP_Select.setAI_START_Polarity(theSTC->AI_START_STOP_Select.
        kAI_START_PolarityActive_High_Or_Rising_Edge);
298   theSTC->AI_START_STOP_Select.flush();
299
300   theSTC->AI_Mode_2.setAI_SI_Initial_Load_Source(theSTC->AI_Mode_2.
        kAI_SI_Initial_Load_SourceLoad_A);
301   theSTC->AI_Mode_2.setAI_SI_Reload_Mode(theSTC->AI_Mode_2.
        kAI_SI_Reload_ModeNo_Change);
302   theSTC->AI_SI_Load_A.writeRegister(ConfigADQ->ScanInterval);
303   theSTC->AI_Command_1.writeAI_SI_Load(1);
304
305   theSTC->Joint_Reset.setAI_Configuration_Start(0);
306   theSTC->Joint_Reset.setAI_Configuration_End(1);
307   theSTC->Joint_Reset.flush();
308
309   return;
310 }
311
312
313
314 void BoardAPI_EndOfScan(tSTC *theSTC, s_config_adq *ConfigADQ) {
315
316   theSTC->Joint_Reset.writeAI_Configuration_Start(1);
317
318   theSTC->AI_START_STOP_Select.setAI_STOP_Select(theSTC->AI_START_STOP_Select.
        kAI_STOP_SelectIN );     //We basically never stop??
319   theSTC->AI_START_STOP_Select.setAI_STOP_Edge(0);
320   theSTC->AI_START_STOP_Select.setAI_STOP_Polarity(theSTC->AI_START_STOP_Select.
        kAI_STOP_PolarityActive_High_Or_Rising_Edge);
321   theSTC->AI_START_STOP_Select.setAI_STOP_Sync(1);
322   theSTC->AI_START_STOP_Select.flush();
323
324   theSTC->Joint_Reset.setAI_Configuration_Start(0);
```

```
325   theSTC->Joint_Reset.setAI_Configuration_End(1);
326   theSTC->Joint_Reset.flush();
327
328   return;
329 }
330
331
332
333 void BoardAPI_ConvertSignal(tSTC *theSTC) {
334
335   theSTC->Joint_Reset.writeAI_Configuration_Start(1);
336
337   theSTC->AI_Mode_2.writeAI_SI2_Initial_Load_Source(theSTC->AI_Mode_2.
        kAI_SI2_Initial_Load_SourceLoad_A);
338   theSTC->AI_Mode_2.writeAI_SI2_Reload_Mode(theSTC->AI_Mode_2.
        kAI_SI2_Reload_ModeNo_Change);
339   theSTC->AI_SI2_Load_A.writeRegister(0xA0);
340   theSTC->AI_Command_1.writeAI_SI2_Load(1);
341
342   theSTC->Joint_Reset.setAI_Configuration_Start(0);
343   theSTC->Joint_Reset.setAI_Configuration_End(1);
344   theSTC->Joint_Reset.flush();
345   return;
346 }
347
348
349
350 void BoardAPI_Arming(tSTC *theSTC) {
351
352   theSTC->AI_Command_1.setAI_SC_Arm(1);
353   theSTC->AI_Command_1.setAI_SI_Arm(1);
354   theSTC->AI_Command_1.setAI_SI2_Arm(1);
355   theSTC->AI_Command_1.setAI_DIV_Arm(1);
356   theSTC->AI_Command_1.flush();
357   return;
358 }
359
360
361
362 void BoardAPI_StartTheAcquisition(tSTC *theSTC) {
363
364   theSTC->AI_Command_2.writeAI_START1_Pulse(1);
365   return;
366 }
```

C

Simulink S-Function Programming

For all the experiments performed during this work C-coded Simulink S-Functions had to be made in order to perform the following tasks:

- Read the ten 12-bit sEMG channels coming from the NI card.
- Buffer the desired data in the system's SDRAM for latter dumping into the hard drive.
- Obtain the muscle activation factor and liberalize it to the system's requirements.

C.1 NI PCI card reader

Once the driver for the card was written an S-Function had to be written which could retrieve the data from the device node placed in the directory /dev and make them available to the model. Since the device driver developed has many configuration arguments when it is launched those arguments will also have to be included in the S-Function in order for this latter one to be able to retrieve the data from the device node correctly.

In Figure C.1 the simulink blok can be seen and in Figure C.2 the configuration mask for the block is shown, where all the configuration parameters are explained again.

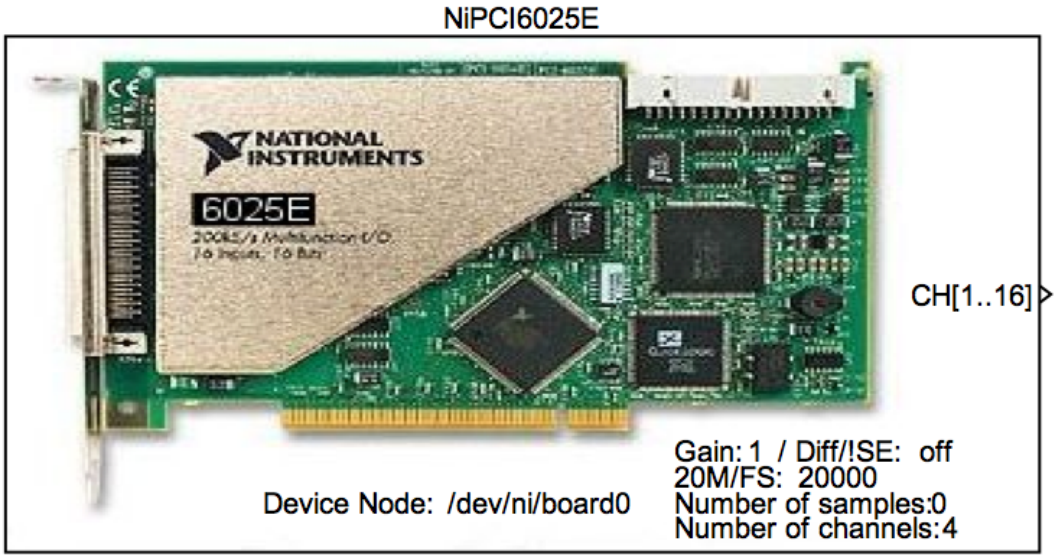

Figure C.1: Here the Simulink block for the S-Function can be seen. As it can be observed the current configuration appears on top of the block and the output bus automatically re-sizes to fit the specified number of channels.

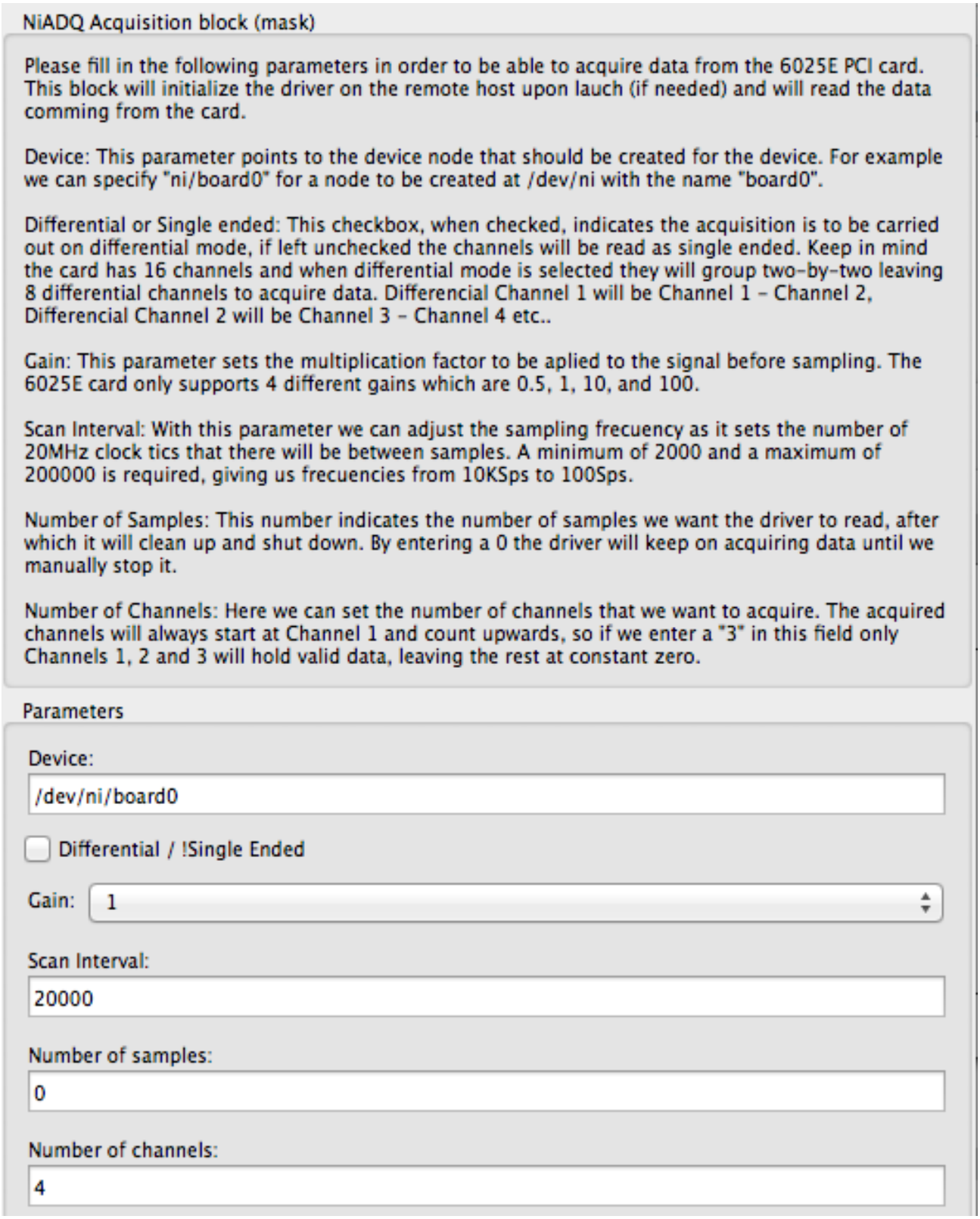

Figure C.2: Here the configuration mask for the simulink block of the S-Function can be seen. As shown a brief description of all configuration parameters is present.

Next the code for the S-Function is presented. This code is in charge of reading a set of samples (more or less depending on the number of channels configured) and making them available to the output ports. This C code was then compiled for the target platform, QNX Neutrino 6.3 along with the Simulink RealTime WorkShop.

Listing C.1: buffer.h

```
#define S_FUNCTION_NAME  sfun_NiADQ
#define S_FUNCTION_LEVEL 2


#include <stdio.h>
#include "NiADQ_Data.h"
#include "simstruc.h"
#include "simstruc_types.h"

#ifdef  MATLAB_MEX_FILE
#else
    #include <libc.h>
#endif

int fid;
int byte_count;
s_config_adq     Command;
s_sample         NewSample;


extern "C" {

#define MDL_CHECK_PARAMETERS   /* Change to #undef to remove function */
#if defined(MDL_CHECK_PARAMETERS)

static void mdlCheckParameters(SimStruct *S) {

    if (mxGetScalar(ssGetSFcnParam(S,1)) < 0 || mxGetScalar(ssGetSFcnParam(S,1)) > 1) {
        ssSetErrorStatus(S,"Differential or Single Ended parameter should be either 1 or 0.");
        return;
    }
    if (mxGetScalar(ssGetSFcnParam(S,3)) < 2000 || mxGetScalar(ssGetSFcnParam(S,3)) > 2000000) {
        ssSetErrorStatus(S,"Scan Iterval parameter should be between 2000 and 200000.");
        return;
    }
    if (mxGetScalar(ssGetSFcnParam(S,4)) < 0) {
        ssSetErrorStatus(S,"Number of Samples should not be negative.");
        return;
    }
    if (mxGetScalar(ssGetSFcnParam(S,5)) < 1 || (mxGetScalar(ssGetSFcnParam(S,1)) == 1 && mxGetScalar(ssGetSFcnParam(S,5)) > 8) || (mxGetScalar(ssGetSFcnParam(S,1)) == 0 && mxGetScalar(ssGetSFcnParam(S,5)) > 16))  {
        ssSetErrorStatus(S,"Number of Channels must be between 1 and 16 for single ended mode and between 1 and 8 in differential mode.");
        return;
```

```
43      }
44      return;
45  }
46  #endif /* MDL_CHECK_PARAMETERS */
47
48
49
50
51  static void mdlInitializeSizes(SimStruct *S) {
52
53      int_T nInputPorts  = 0;
54      int_T nOutputPorts = 1;
55
56      int_T inputPortIdx  = 0;
57      int_T outputPortIdx = 0;
58
59      ssSetNumSFcnParams(S, 6);
60      if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) { return; }
61      #ifdef MDL_CHECK_PARAMETERS
62      mdlCheckParameters(S);
63      if (ssGetErrorStatus(S) != NULL) return;
64      #endif
65    /*ssSetSFcnParamTunable(S, 0, 0);*/
66
67      if (!ssSetNumInputPorts(S, nInputPorts)) return;
68      if (!ssSetNumOutputPorts(S, nOutputPorts)) return;
69
70      ssSetOutputPortWidth(S, 0, (int) mxGetScalar(ssGetSFcnParam(S,5)));
71
72     // ssSetNumPWork(S, 1);
73     // ssSetNumIWork(S, 1);
74
75    //  ssSetOptions( S, SS_OPTION_EXCEPTION_FREE_CODE |
      SS_OPTION_DISCRETE_VALUED_OUTPUT | SS_OPTION_PLACE_ASAP);   /* general options (
      SS_OPTION_xx)          */
76      ssSetOptions( S, SS_OPTION_EXCEPTION_FREE_CODE);
77  }
78
79
80
81  static void mdlInitializeSampleTimes(SimStruct *S) {
82
83      ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
84      ssSetOffsetTime(S, 0, 0.0);
85      ssSetModelReferenceSampleTimeDefaultInheritance(S);
86
87  }
88
89
90  #define MDL_START
91  static void mdlStart(SimStruct *S) {
92
93    #if defined(RT)
94    char  Temp[10], Device_Node[100];
95    int   gain, i;
96
```

```
97    ssPrintf("\n\n\r========== NiPCI6025E block initialization parameters ==========\n\r"
        );
98
99    mxGetString(ssGetSFcnParam(S,0), Device_Node, 100);
100   Command.Diff = (int) mxGetScalar(ssGetSFcnParam(S,1));
101   mxGetString(ssGetSFcnParam(S,2), Temp, 10);
102   if (!strcmp("0.5", Temp)) { Command.Gain = 1; }
103   if (!strcmp("1", Temp))   { Command.Gain = 2; }
104   if (!strcmp("10", Temp))  { Command.Gain = 3; }
105   if (!strcmp("100", Temp)) { Command.Gain = 4; }
106   Command.ScanInterval =  (int) mxGetScalar(ssGetSFcnParam(S,3));
107   Command.NumberOfSamples = (int) mxGetScalar(ssGetSFcnParam(S,4));
108   Command.NumberOfChannels = (int) mxGetScalar(ssGetSFcnParam(S,5));
109
110   ssPrintf("\n\rThe parameters are the following:");
111   ssPrintf("\n\r\tDevice Node: %s", Device_Node);
112   ssPrintf("\n\r\tDiff/!Se: %d", Command.Diff);
113   ssPrintf("\n\r\tGain: %d", Command.Gain);
114   ssPrintf("\n\r\tScan Interval: %d", Command.ScanInterval);
115   ssPrintf("\n\r\tNumber of samples: %d", Command.NumberOfSamples);
116   ssPrintf("\n\r\tNumber of channels: %d", Command.NumberOfChannels);
117
118   fid = open(Device_Node, O_RDWR);
119
120   if (fid <= 0) {
121       ssPrintf("\n\rError trying to launch de driver, for some reason %s could not
        be opened. FID = %d\n\r", Device_Node, fid);
122       ssSetErrorStatus(S,"Error trying to launch de driver, for some reason %s could
       not be opened");
123       return;
124   } else { ssPrintf("\n\rNode file opened succesfully! FID = %d", fid); }
125
126   i = write(fid, &Command, sizeof(s_config_adq));
127   if (i == sizeof(s_config_adq)) { ssPrintf("\n\rCommand written successfully"); }
128   else { ssPrintf("\n\rError trying to write the command to the driver"); }
129
130   byte_count = 0;
131
132  // ssSetPWorkValue(S, 0, (void *) &NewSample);
133  // ssSetIWorkValue(S, 1, (int) fid);
134
135   ssPrintf("\n\n\r========== NiPCI6025E block initialization done! ==========\n\n\r");
136
137  #endif
138 }
139
140
141 static void mdlOutputs(SimStruct *S, int_T tid) {
142
143    #if defined(RT)
144
145     int i;
146    // s_sample *Sample = (s_sample*) ssGetPWorkValue(S,0);
147    // int f = (int) ssGetIWorkValue(S,1);
148     real_T * y = ssGetOutputPortRealSignal(S, 0);
149
```

```
150     byte_count =+ read(fid, &(NewSample.Channels[byte_count/2]), sizeof(s_sample) -
        byte_count);
151
152     if (byte_count == sizeof(s_sample)) {
153
154         byte_count = 0;
155         for( i = 0 ; i <  (int) mxGetScalar(ssGetSFcnParam(S,5)) ; i++ ) {
156             y[i] = (double) NewSample.Channels[i];
157         }
158
159     }
160
161     #endif
162
163 }
164
165
166 static void mdlTerminate(SimStruct *S) {
167
168     #if defined(RT)
169
170     int      i, a;
171     char     kill_string[] = "slay -fQ DD_NiADQ";
172    // int      f = (int) ssGetIWorkValue(S,1);
173
174     close(fid);
175     //system(kill_string);
176
177     #endif
178
179 }
180
181 }
182
183
184 /*=============================*
185  * Required S-function trailer *
186  *=============================*/
187
188 #ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
189 #include "simulink.c"      /* MEX-file interface mechanism */
190 #else
191 #include "cg_sfun.h"       /* Code generation registration function */
192 #endif
```

C.2 QNX Data Logger

A characteristic of Real-Time operating systems is that if a certain process takes too long to perform it's required operations the system's dispatcher stops such process and carries on with the other pending tasks. This became a problem when large amounts of data where to be recorded and they could not be sent over TCP/IP during the execution. At

first the model was intended to write directly to the hard drive, but after a few tries it became clear that reading/writing that amount of data to a hard drive would take more than 1 millisecond (the system's step time) so the model would repeatedly stop working. This was at first hard to diagnose because it would happen at a relative random time, since even if a process asks to write to a file, the OS can choose to buffer that data in SDRAM and then eventually begin a dump to the HD. It is for this reason that the failure would not occur right at the start of the execution and not always at the same time after launch.

The solution for this was to write another S-Function that would reserve a certain amount of SDRAM and then write to it in a ring-buffer manner while only writing it to the file in the HD upon specific request and by launching a different thread. Figure C.3 shows the S-Function's Simulink block and Figure C.4 the block's configuration mask.

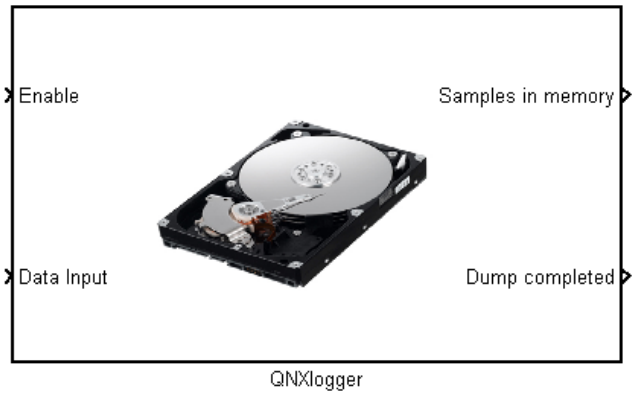

Figure C.3: Here the Simulink block for the QNX Data-Logger S-Function can be seen. All the port's functions are explained in the configuration mask show in Figure C.4

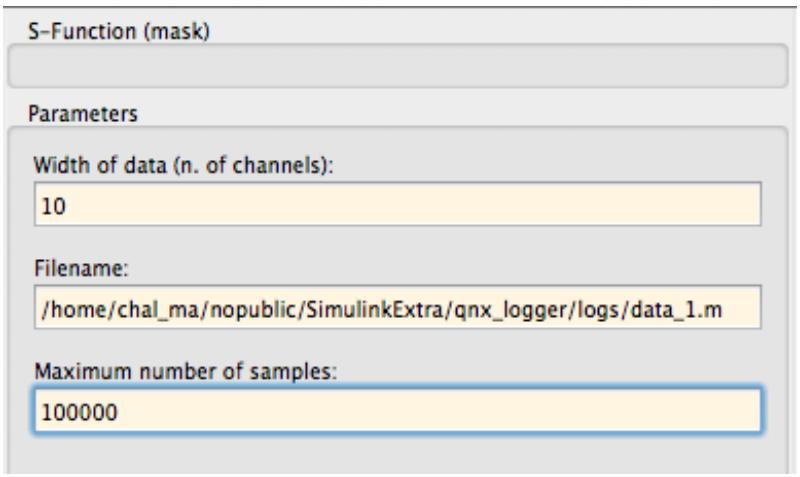

Figure C.4: Here the configuration mask for the simulink block of the S-Function can be seen. As shown a brief description of all configuration parameters is present.

Next the code for the S-Function is presented. This code is in charge of reserving the memory pool, saving the data to it, and then, upon request, launch a process which writes the buffer to a file. This C code was then compiled for the target platform, QNX Neutrino 6.3 along with the Simulink RealTime WorkShop.

Listing C.2: buffer.h

```
#ifndef BUFFER_H
#define BUFFER_H

struct Buffer{
    int max_samples;
    int write_width;
    int last_index;
    int looped;
    float** data;
};

#endif
```

Listing C.3: logblock.h

```
#ifndef LOG_BLOCK_H
#define LOG_BLOCK_H


#include <pthread.h>
#include <stdio.h>
#include <string>
#include <cstdlib>
#include <iostream>

#include "buffer.h"

void * write_to_file(void * par);

class Logblock {

  public:
    Buffer write_buf;
    pthread_mutex_t lock_buf_write;
    char write_name[255];
    pthread_t thread_write;
    char file_saved;
    int start_write(Logblock* self,char name[]);
    Logblock(int write_width, int max_samples);
    ~Logblock();
};

#endif // LOG_BLOCK_H
```

Listing C.4: sfun_QNXlogger.cpp

```
#include <stdio.h>
#include <cstdlib>
#include <iostream>
```

```
4  #include <string>
5  #include <unistd.h>
6
7
8  #ifdef  MATLAB_MEX_FILE
9  #else
10   #include <libc.h>
11 #endif
12
13 #include "logblock.h"
14
15 #define S_FUNCTION_LEVEL 2
16 #define S_FUNCTION_NAME  sfun_QNXlogger
17
18 #include "simstruc.h"
19
20 #define IS_PARAM_DOUBLE(pVal) (mxIsNumeric(pVal) && !mxIsLogical(pVal) &&\
21 !mxIsEmpty(pVal) && !mxIsSparse(pVal) && !mxIsComplex(pVal) && mxIsDouble(pVal))
22
23
24 struct Params{
25     int max_samples;
26     int width_data;
27     int current_index;
28     int old_enable;
29     char filename[1024];
30 };
31
32 extern "C" {
33
34 static void mdlInitializeSizes(SimStruct *S) {
35
36    ssSetNumSFcnParams(S, 3);
37    // buffer_size and data_width are not tuname
38    ssSetSFcnParamTunable( S, 0, SS_PRM_NOT_TUNABLE );
39    ssSetSFcnParamTunable( S, 1, SS_PRM_NOT_TUNABLE );
40
41    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) return;
42    if (!ssSetNumInputPorts(S, 2)) return;
43    if (!ssSetNumOutputPorts(S, 2)) return;
44
45    ssSetInputPortWidth(S, 0, 1);  // on/off logging
46    ssSetInputPortWidth(S, 1, (int)(mxGetScalar(ssGetSFcnParam(S, 1))));// data to
       log
47    ssSetOutputPortWidth(S, 0, 1);  // number of samples
48    ssSetOutputPortWidth(S, 1, 1);  // dump completed
49
50    ssSetInputPortDirectFeedThrough(S, 0, 1);
51    ssSetInputPortDirectFeedThrough(S, 1, 1);
52
53    ssSetNumSampleTimes(S, 1);
54
55    ssSetNumPWork(S, 2);
56
57    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
58 }
59
```

```
60
61 static void mdlInitializeSampleTimes(SimStruct *S) {
62
63     ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
64     ssSetOffsetTime(S, 0, 0.0);
65     ssSetModelReferenceSampleTimeDefaultInheritance(S);
66  }
67
68
69
70 #define MDL_START
71 static void mdlStart(SimStruct *S) {
72
73     #if defined(RT)
74     ssPrintf("\n========== RT logging block  mdlStart ==========\n");
75
76    Params * Parameters = new Params();
77
78    Parameters->max_samples =  (int) (mxGetScalar(ssGetSFcnParam(S, 0)));
79    Parameters->width_data =  (int) (mxGetScalar(ssGetSFcnParam(S, 1)));
80    mxGetString(ssGetSFcnParam(S,2), Parameters->filename, 1024);
81    Parameters->current_index = 0;
82    Parameters->old_enable = 0;
83
84    ssPrintf("\n\rBuffer Size in samples (1 sample = 1double x data_width): %d",
       Parameters->max_samples);
85    ssPrintf("\n\rData width (in doubles): %d", Parameters->width_data);
86    ssPrintf("\n\rInitial filename: %s", Parameters->filename);
87    ssPrintf("\n\rMemory requested: max_samples * width_data = %d floats", Parameters
       ->width_data*Parameters->max_samples);
88
89    Logblock * Logger = new Logblock( Parameters->width_data , Parameters->
       max_samples ) ;
90
91    pthread_mutex_init(&(Logger->lock_buf_write), NULL);
92
93    ssGetPWork(S)[0] =(void *) Logger;
94    ssGetPWork(S)[1] =(void *) Parameters;
95
96    real_T *ready = (real_T *)ssGetOutputPortSignal(S,1);
97    real_T *samples_in_memory = (real_T *)ssGetOutputPortSignal(S,0);
98
99    ready[0] = 1;
100   samples_in_memory[0] = 0;
101
102    ssPrintf("\n\n========== RT logging block  mdlStart : OK \n\n");
103
104    #endif
105
106 }
107
108
109 static void mdlOutputs(SimStruct *S, int_T tid) {
110
111    #if defined(RT)
112
113
```

```
114     #endif
115 }
116
117
118
119 #define MDL_UPDATE
120 static void mdlUpdate(SimStruct *S, int_T tid) {
121
122     #if defined(RT)
123
124     Params * Parameters = (Params*) ssGetPWorkValue(S,1);
125     Logblock * Logger =  (Logblock*) ssGetPWorkValue(S,0);
126     real_T *ready = (real_T *)ssGetOutputPortSignal(S,1);
127
128     InputRealPtrsType enable = ssGetInputPortRealSignalPtrs(S, 0);
129
130     if((int) (*enable[0]) == 1){
131
132         int res = pthread_mutex_trylock(&(Logger->lock_buf_write));
133
134         if (res == 0) {
135
136             ready[0] = 0;
137
138             int width = Parameters->width_data;
139             int index = Parameters->current_index;
140             InputRealPtrsType data = ssGetInputPortRealSignalPtrs(S, 1);
141
142             for( int j = 0 ; j < width ; j++) {
143                 Logger->write_buf.data[j][index] = (float) (*data[j]);
144             }
145
146             index++;
147             real_T *samples_in_memory = (real_T *)ssGetOutputPortSignal(S,0);
148             if (Logger->write_buf.looped == 1) {
149                 samples_in_memory[0] = Logger->write_buf.max_samples;
150             } else {
151                 samples_in_memory[0] = index - 1;
152             }
153
154             if ( index >= Logger->write_buf.max_samples) {
155                 Logger->write_buf.looped = 1;
156                 index = 0;
157             }
158
159             Logger->write_buf.last_index = index;
160             Parameters->current_index = index;
161
162             pthread_mutex_unlock(  &(Logger->lock_buf_write)  );
163
164         }
165     }
166
167
168     if ((int)(*enable[0]) == 0) {
169
170         int res;
```

```
171
172         if (Parameters->old_enable == 0 && ready[0] == 0) {
173
174             res = pthread_mutex_trylock(&(Logger->lock_buf_write));
175
176             if (res == 0) {
177
178                 ssPrintf("\n\r>Logger: No data left in buffer to be dumped, we seem
    to be done.\n\n");
179                 Logger->write_buf.last_index = 0;
180                 Parameters->current_index = 0;
181                 Logger->write_buf.looped = 0;
182                 real_T *samples_in_memory = (real_T *)ssGetOutputPortSignal(S,0);
183                 samples_in_memory[0] = 0;
184                 ready[0] = 1;
185                 pthread_mutex_unlock(  &(Logger->lock_buf_write)  );
186
187             }
188         }
189
190
191         if (Parameters->old_enable == 1) {
192
193             ssPrintf("\n\n\r>Logger: Launching dumping process...");
194
195             mxGetString(ssGetSFcnParam(S,2), Parameters->filename, 1024);
196
197             Logger->write_buf.last_index = Parameters->current_index;
198             Logger->start_write(Logger, Parameters->filename);
199
200             Parameters->current_index = 0;
201         }
202     }
203
204     Parameters->old_enable = (int) (*enable[0]) ;
205
206     #endif
207 }
208
209
210
211 static void mdlTerminate(SimStruct *S) {
212
213     #if defined(RT)
214     Logblock * Logger =  (Logblock*) ssGetPWorkValue(S,0);
215     Params* Parameters =  (Params*) ssGetPWorkValue(S,1);
216     #endif
217 }
218
219 }
220
221
222 #ifdef  MATLAB_MEX_FILE     /* Is this file being compiled as a MEX-file? */
223 #include "simulink.c"      /* MEX-file interface mechanism */
224 #else
225 #include "cg_sfun.h"       /* Code generation registration function */
226 #endif
```

C.3 Stiffness Estimator

Once the training data had been gathered and the dimensionality of the sEMG values reduced to one single muscle activation factor a Simulink block was needed in order to be able to perform the transformation from the incoming sEMG data to this muscle activation value. This block is given the principal component or Eigenvector and a few more parameter and then it generates, in real time, the stiffness to be fed to the controller based on the current sEMG values.

In Figure C.5 the Simulink block can be seen and in Figure C.6 it's configuration mask is shown.

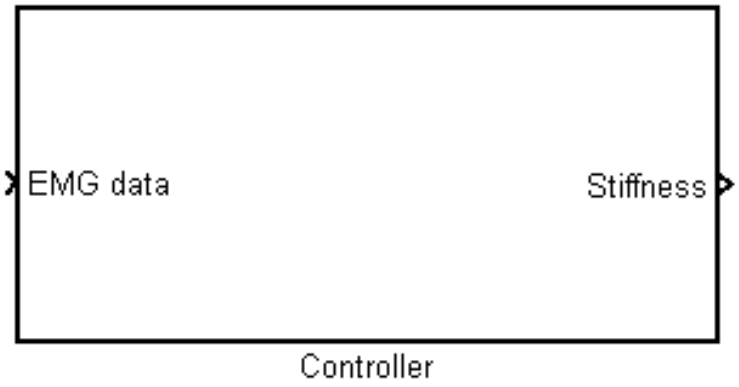

Figure C.5: Here the Simulink block for the S-Function can be seen. As it can be seen it's operation is pretty simple, as once configured through the mask, it only needs the current sEMG values.

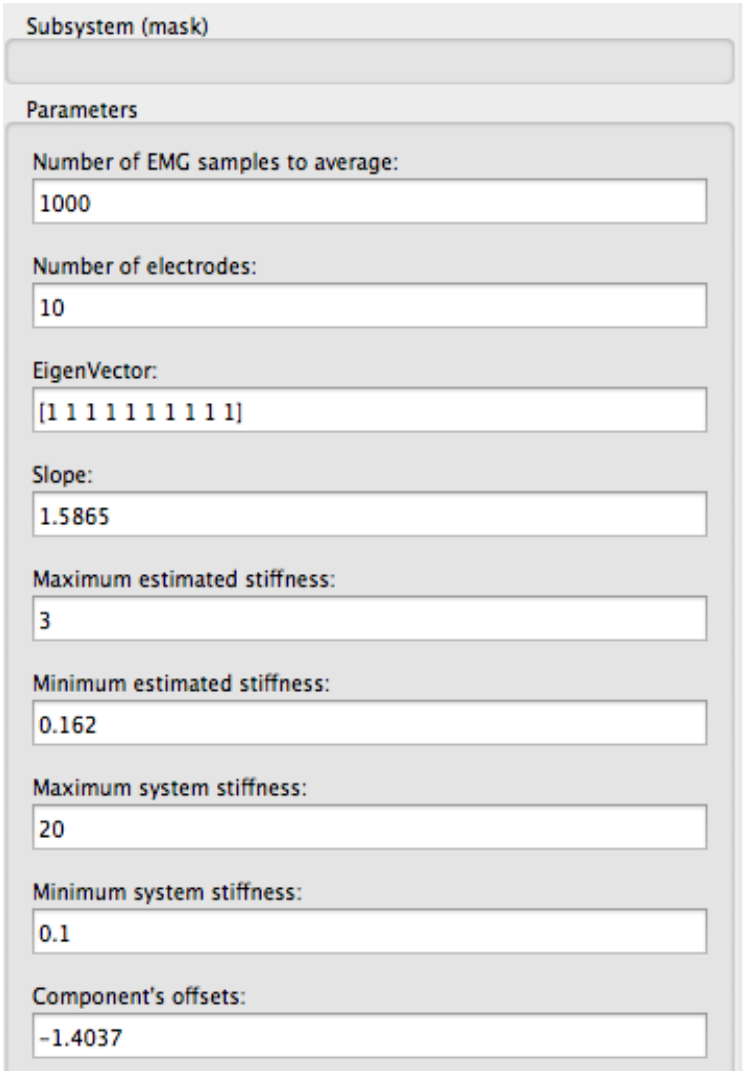

Figure C.6: Here the configuration mask for the simulink block of the S-Function can be seen. The information with which this block is filled is extracted from the training data.

Next the code for the S-Function is presented. This code is in charge of reducing the sEMG values to a muscle activation factor using the provided Eigenvector and then linearize and saturate the output stiffness to fit the dynamic range of the controller. This C code was then compiled for the target platform, QNX Neutrino 6.3 along with the Simulink RealTime WorkShop.

Listing C.5: sfun_QNXlogger.cpp

```
1 #define S_FUNCTION_NAME  sfun_EMG_to_Stiffness_1Pos
2 #define S_FUNCTION_LEVEL 2
3
4
5 #include <stdio.h>
6 #include "simstruc.h"
7 #include "simstruc_types.h"
8
```

```
9
10 #ifdef  MATLAB_MEX_FILE
11 #else
12     #include <libc.h>
13 #endif
14
15
16 extern "C" {
17
18 #define MDL_CHECK_PARAMETERS   /* Change to #undef to remove function */
19 #if defined(MDL_CHECK_PARAMETERS)
20
21 static void mdlCheckParameters(SimStruct *S) {
22
23     if (mxGetScalar(ssGetSFcnParam(S,0)) < 0 || mxGetScalar(ssGetSFcnParam(S,0)) >
    16) {
24         ssSetErrorStatus(S,"Invalid value for the number of electrodes, it must be
    between 1 and 16.");
25         return;
26     }
27     if (mxGetNumberOfElements(ssGetSFcnParam(S,2)) != mxGetScalar(ssGetSFcnParam(S
    ,0))) {
28         ssSetErrorStatus(S,"The number of components in the eigenvector should be
    equal to the number of electrodes");
29         return;
30     }
31     if (mxGetScalar(ssGetSFcnParam(S,4)) < mxGetScalar(ssGetSFcnParam(S,5))) {
32         ssSetErrorStatus(S,"The maximum estimated stiffness must be equal or larger
    than the minimum stiffness");
33         return;
34     }
35     if (mxGetScalar(ssGetSFcnParam(S,6)) < mxGetScalar(ssGetSFcnParam(S,7))) {
36         ssSetErrorStatus(S,"The maximum stiffness must be equal or larger than the
    minimum stiffness");
37         return;
38     }
39     return;
40 }
41 #endif /* MDL_CHECK_PARAMETERS */
42
43
44
45
46 static void mdlInitializeSizes(SimStruct *S) {
47
48     int_T nInputPorts  = 1;
49     int_T nOutputPorts = 1;
50
51     int_T inputPortIdx  = 0;
52     int_T outputPortIdx = 0;
53
54     ssSetNumSFcnParams(S, 8);
55     if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) { return; }
56     #ifdef MDL_CHECK_PARAMETERS
57     mdlCheckParameters(S);
58     if (ssGetErrorStatus(S) != NULL) return;
59     #endif
```

```
60    /*ssSetSFcnParamTunable(S, 0, 0);*/
61
62      if (!ssSetNumInputPorts(S, nInputPorts)) return;
63      if (!ssSetNumOutputPorts(S, nOutputPorts)) return;
64
65      ssSetOutputPortWidth(S, 0, 1);
66      ssSetInputPortWidth(S, 0, (int)mxGetScalar(ssGetSFcnParam(S,0)));
67
68      ssSetInputPortDirectFeedThrough(S, 0, 1);
69
70      ssSetNumSampleTimes(S, 1);
71
72     // ssSetNumIWork(S, 1);
73
74    //  ssSetOptions( S, SS_OPTION_EXCEPTION_FREE_CODE |
       SS_OPTION_DISCRETE_VALUED_OUTPUT | SS_OPTION_PLACE_ASAP);   /* general options (
       SS_OPTION_xx)          */
75      ssSetOptions( S, SS_OPTION_EXCEPTION_FREE_CODE);
76  }
77
78
79
80  static void mdlInitializeSampleTimes(SimStruct *S) {
81
82      ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
83      ssSetOffsetTime(S, 0, 0.0);
84      ssSetModelReferenceSampleTimeDefaultInheritance(S);
85
86  }
87
88
89  #define MDL_START
90  static void mdlStart(SimStruct *S) {
91
92    #if defined(RT)
93
94      ssPrintf("\n========== EMG based stiffness controller block  mdlStart ==========\n")
         ;
95      ssPrintf("\n================================== Done ==================================\n\n
         \n");
96    #endif
97  }
98
99
100 static void mdlOutputs(SimStruct *S, int_T tid) {
101
102     #if defined(RT)
103
104
105     #endif
106 }
107
108
109 #define MDL_UPDATE
110 static void mdlUpdate(SimStruct *S, int_T tid) {
111
112     #if defined(RT)
```

```
113
114     float K_est = 0;
115     float slope = mxGetScalar(ssGetSFcnParam(S,3));
116     float max_est_k = mxGetScalar(ssGetSFcnParam(S,4));
117     float min_est_k = mxGetScalar(ssGetSFcnParam(S,5));
118     float max_sys_k = mxGetScalar(ssGetSFcnParam(S,6));
119     float min_sys_k = mxGetScalar(ssGetSFcnParam(S,7));
120     int i;
121
122     InputRealPtrsType Electrodes  = ssGetInputPortRealSignalPtrs(S,0);
123     real_T *Stiffness = (real_T *)ssGetOutputPortSignal(S, 0);
124
125     int_T vectlen = mxGetScalar(ssGetSFcnParam(S,0));
126     int_T Offset = mxGetScalar(ssGetSFcnParam(S,1));
127     const real_T *EigVec = mxGetPr(ssGetSFcnParam(S,2));
128
129
130     for ( i = 0 ; i < vectlen ; i++ ) {
131
132         K_est += (*Electrodes[i])*(EigVec[i]);
133     }
134
135     K_est = K_est * slope;
136     K_est += Offset;
137
138     K_est = (K_est - min_est_k)/max_est_k;
139
140     if (K_est > 1) {
141         K_est = 1;
142     }
143     if (K_est < 0) {
144         K_est = 0;
145     }
146
147     Stiffness[0] = (K_est)*(max_sys_k - min_sys_k) + min_sys_k;
148
149     #endif
150
151 }
152
153
154 static void mdlTerminate(SimStruct *S) {
155
156     #if defined(RT)
157
158
159     //system(kill_string);
160
161     #endif
162
163 }
164
165 }
166
167
168 /*=============================*
169  * Required S-function trailer *
```

```
170  *=================================*/
171
172 #ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
173 #include "simulink.c"      /* MEX-file interface mechanism */
174 #else
175 #include "cg_sfun.h"       /* Code generation registration function */
176 #endif
```

D

Otto-Bock Differential sEMG Electrodes

To capture the sEMG signals Otto-Bock 13E200=50 electrodes were used. These electrodes are differential (they have three terminals and therefore are capable of capturing MU activity without any other reference signal) and active, meaning that they have a built-in amplifier which generates a $\pm 5V$ at the electrodes output. This early amplification stage minimizes ambient noise coupling improving the final SNR ratio and therefore the sensibility of the signal captured. Furthermore, and as mentioned in this work, this electrodes do not provide the raw sEMG signal, but an envelope of such. Let's keep in mind that this process, similar to a demodulation, is not an LTI operation. For more information from the manufacturer of the equipment please see [15].

In order to validate the delay introduced by the envelope detection and filtering further work had been done at the Deutches Zentrum f§r Luft-und-Raumfahrt, RMC, Obberpfaffenhofen. Next the result of such tests are presented. Credits to Dominic Lakatos, a current researcher at RMC.

One of the differential inputs (contacts: middle, cable output side) of the Otto Bock 13E200=50 EMG surface electrode was connected to an analog signal generator (KROHN HITE, Model 2400-1). Settings:

- waveform: sine
- frequency multiplier: 100 and 1k
- peak (in Volts): 5, attenuator: 60dB

The amplifiere level of the eletrodes was set to "'5"'. Both the input voltage $U_i(t)$ and the output voltage $U_o(t)$ were recorded by a NIDAQmx PCI6023E on real-time OS QNX with a samplerate of 10 kHz. The frequency response was identified by adjusting a frequency and measuring $U_o(t)$ in a stationary state. Notice: the EMG electrode behaves not like a LTI.

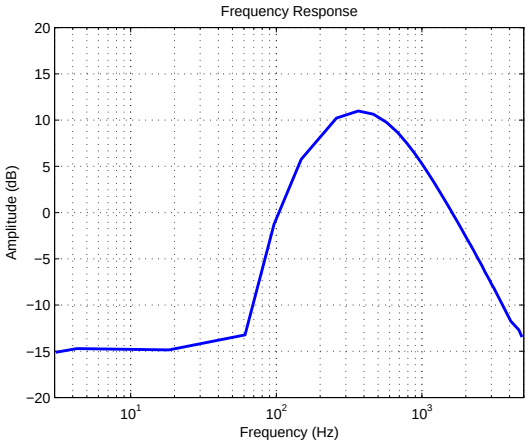

Figure 1: A $= 20\log(\bar{U}_o)$, $\bar{U}_o = \frac{1}{T}\int_0^T U_o(t)dt$

The delay of the electrode was measured by manually switching on the signal generator.

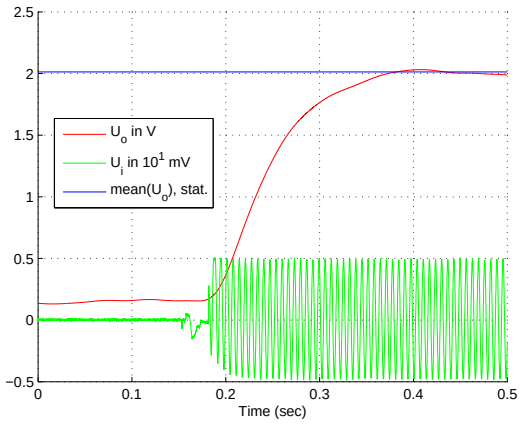

Figure 2: typical plot for delay measurements

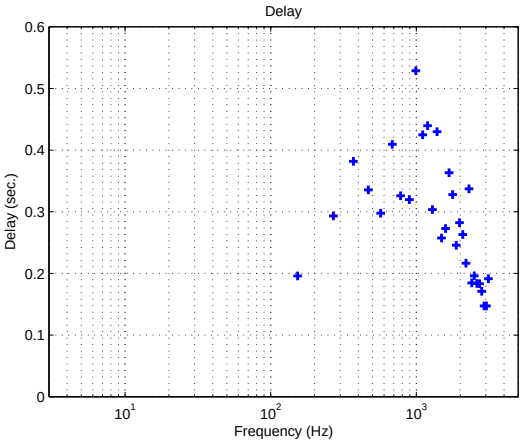

Figure 3: delay over frequency

Bibliography

[1] A. Ajoudani, M. Gabiccini, NG. Tsagarakis, A. Albu-Schaeffer, and A. Bicchi. Teleimpedance: Exploring the role of common-mode and configuration-dependant stiffness. *IEEE-RAS International Conference on Humanoid Robots*, pages 363–369, 2012.

[2] Arash Ajoudani, Nikos Tsagarakis, and Antonio Bicchi. Tele-impedance: Teleoperation with impedance regulation using a bodyâĹŠmachine interface. *The International Journal of Robotics Research*, 31(13):1642–1655, October 2012.

[3] Leandro Ricardo Altimari, José Luiz Dantas, Marcelo Bigliassi, Thiago Ferreira Dias Kanthack, Antonio Carlos de Moraes, and Taufik Abrão. *Computational Intelligence in Electromyography Analysis - A Perspective on Current Applications and Future Challenges*, chapter 5. InTech, 1 edition, 2012.

[4] Jordi Artigas, Jee-Hwan Ryu, Carsten Preusche, and Gerd Hirzinger. Network representation and passivity of delayed teleoperation systems. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 177–183, 2011.

[5] Dr. Scott Day. *Important Factors in Surface EMG Measurement.* Bortec Biomedical. [Online: `http://www.andrewsterian.com/courses/214/EMG_measurement_and_recording.pdf` accessed 2-September-2013].

[6] Ildar Farkhatdinov (1) Ildar Farkhatdinov and Jee-Hwan Ryu. *Haptics: Perception, Devices and Scenarios*, volume 13, 6th international eurohaptics conference, proceedings 5, pages 796–805. Springer, 2008.

[7] David W. Franklin, Gary Liaw, Theodore E. Milner, Rieko Osu, Etienne Burdet, and Mitsuo Kawato. Endpoint stiffness of the arm is directionally tuned to instability in the environment. *The Journal of Neuroscience*, 27(29):7705–7716, July 2007.

[8] Blake Hannaford and Jee-Hwan Ryu. Time-domain passivity control of haptic interfaces. *IEEE Transactions on Robotics and Automation (ICRA)*, 18(1):1–10, 2002.

[9] N. Hogan. Impedance control: An approach to manipulation. *American Control Conference*, pages 304–313, June 1984.

[10] Dominic Lakatos, Daniel Rüschen, Justin Bayer, Jörn Vogel, and Patrick van der Smagt. Identification of human limb stiffness in 5 dof and estimation via emg. *Springer Tracts in Advanced Robotics*, 88:89–99, 2013.

[11] Frank H. Netter MD. *Atlas of Human Anatomy*. Saunders, 5 edition, 2010.

[12] Roberto Merletti, Matteo Aventaggiato, Alberto Botter, Ales Holobar, Hamid Marateb, and Taian M.M. Vieira. Advances in surface emg: Recent progress in advances in surface emg: Recent progress in detection and processing techniques. *Critical Reviews in Biomedical Engineering*, 38(4):305–345, 2010.

[13] Marvin Minsky. Telepresence. *OMNI magazine*, 1980.

[14] S. Hamid Nawab, Robert P. Wotiz, and Carlo J. De Luca. Decomposition of indwelling emg signals. *Journal of Applied Physiology*, 105(2):700–710, August 2008.

[15] Otto-Bock. *MYOBOCK® Electrodes*. Otto-Bock. [Online: `http://www.ottobock.se/cps/rde/xbcr/ob_se_sv/myobock_8_myobock_electrodes.pdf` accessed 2-September-2013].

[16] H. Parsaei and D.W. Stashuk. Emg signal decomposition using motor unit potential train validity. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 21(2):265–274, March 2013.

[17] Allan Piersol and Thomas Paez. *MECHANICAL IMPEDANCE/MOBILITY*, chapter 10, pages 10.1 – 10.14. McGraw-Hill Professional, 6 edition, 2009.

[18] Jee-Hwan Ryu, Jordi Artigas, and Carsten Preusche. A passive bilateral control scheme for a teleoperator with time-varying communication delay. *Mechatronics*, 20:812–823, 2010.

[19] Milner TE. Contribution of geometry and joint stiffness to mechanical stability of the human arm. *Experimental Brain Research*, 143:515–519, March 2002.

[20] Jörn Vogel, Claudio Castellini, and Patrick van der Smagt. Emg-based teleoperation and manipulation with the dlr lwr-iii. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 672–678, September 2011.

[21] Alan V.Oppenheim, Alan S.Willsky, and S.Hamid. *Signals and Systems*, chapter 6, pages 472–476. Prentice Hall, 1996.

[22] Andrew Whitworth. *Control Systems*, chapter 27, pages 185–188. Wiki-Books, 2013. [Online: `http://upload.wikimedia.org/wikipedia/commons/e/e4/Control_Systems.pdf` accessed 2-September-2013].

[23] Yasuyoshi Yokokohji and Tsuneo Yoshikawa. Bilateral control of master-slave manipulators for ideal kinesthetic bilateral control of master-slave manipulators for ideal kinesthetic bilateral control of master-salve manipulators for ideal kinesthetic coupling-formulation and experiment. *IEEE Transactions on Robotics and Automation*, 10:605–620, October 1994.