BIOROB
EPFL Biorobotics Laboratory

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Semester Project

Microengineering, Master Semester I
September 2010 - January 2011

Design of a demo experimental setup for human augmentation

Author:
Mike Domenik Rinderknecht

Supervisors:
Dr. Renaud Ronsse
Dr. Alessandro Crespi

Professor:
Prof. Auke Jan Ijspeert

Contents

1. Introduction

Modern research on rehabilitation robotics are exploring various ways to cope with the interaction between robots and humans. One of these ways is to avoid forcing the human's movement to a predefined trajectory, and to permit the human the complete control of the device at any moment. Hence, the robot should detect the user's movement intention and react accordingly to assist the movement. New techniques are being developped, which can be through hardware adaptations, e.g. elastic elements between the actuators of the device and the human (Series Elastic Elements, SEE), or by a high level control using adaptive oscillators. The latter is able to synchronize to the human's movement and to estimate the movements parameters (e.g. position, velocity, and acceleration). In consequence, the device can estimate the necessary torque/force to perform the movement and assists the human by a fraction of this torque.
This project is the continuation of the previous work carried out by F. A. Delaloye [1], who designed the mechanical setup used during this semester project. The goals of this project were defined similar to the structure of this report and were very multifarious going from mechanical design to high level control and some experiments with healthy humans. Chapter 2 covers the characterization of the Series Elastic Element. In addition to that, as the safety issue got completely forgotten in [1], there had to be taken some measures. The problem was the high torque that could have been delivered to the human in a situation of malfunction. Thus the torque had to be limited. There exist several methods to limit the torque. Section 2.2 discusses a mechanical solution with a security clutch, Section 4.2 an electrical solution by limitating the current and finally Subsubsection 5.3.2.4 a very high level torque saturation. As the first solution (i.e. the mechanical security clutch) is the safest, but could not be integrated into the existing hardware setup, a new device was designed in Section 2.3. Chapter 3 explains the solutions to the voltage and current problems which could be solved quite easily by adding capacitors and diodes. Chapter 4 deals with the low level control of the device (i.e. the PID control of the motor and the current limitation). Chapter 5 discusses the whole matter of high level control, such as interfacing the device with the computer using MATLAB and Simulink and controlling the device using an adaptive oscillator to assist the human's forearm movement. Chapter 6 contains the performed experiments to investigate the robustness and sensitivity of the approximative models used for the control. This Chapter corresponds to the main part of the publication "Assistance using adaptive oscillators: Sensitivity analysis on the resonance frequency" by Rinderknecht et al., being written in parallel to the project for the International Conference on Rehabilitation Robotics 2011.

2. Mechanics

The hardware setup designed by F. A. Delaloye during a semester project [1] is a simple one degree-of-freedom pendulum, parallel to the participant's forearm, and controlled by a motor through a Series Elastic Element (SEE). Figure 2.1 shows a top view of the assistive device. The SEE introduces compliance and achieves series elastic actuation [2, 3, 4]. The SEE was made of two disks: one coupled to the motor, the other to the pendulum and both connected to each other via six tangential traction springs. The resulting torque provided by the SEE was close to a linear function of the deviation angle between the disks. The positions of the motor and of the participant's elbow are measured by two encoders. Since the participant's forearm was not constrained, the rotation axis of the elbow can be continuously aligned with the one of the assistive device. As consequence, there is no need for additional passive degrees of freedom for axes alignment.

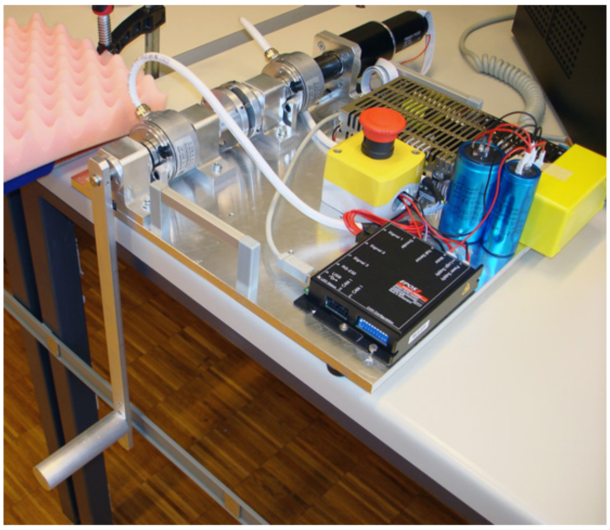


Figure 2.1.: Top view of the assistive device.

2.1. SEE characterization

In order to achieve the highest possible torque with the existing Series Elastic Element (SEE) designed by F. A. Delaloye [1] (see Figure 2.2), the traction springs 08/1/1 from Durovis were chosen.
In this section the identification of the SEE's spring rate will be described. The procedure of the experiment was as follows:

1. Let the pendulum hang vertically down and fix the motor position.

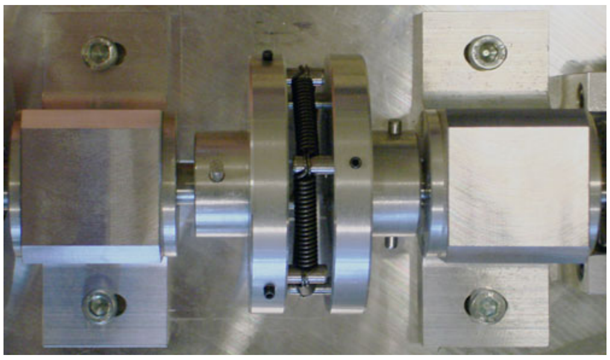


Figure 2.2.: Series Elastic Element.

2. Pull vertically to the pendulum with a dynamometer attached to the end of the pendulum with a certain force F.
3. Measure the angular displacement $\Delta\theta$ of the pendulum with the encoder.

With the values $F = 3N$, $l = 0.33m$ and $\Delta\theta = 830[encoder] = 0.2608rad$ knowing that $20000[encoder] \widehat{=} 2\pi$, the spring rate of the SEE k_{SEE} can be calculated:

$$k_{SEE} = \frac{F \cdot l}{\Delta\theta} = 3.8Nm/rad \tag{2.1}$$

It can be observed that with a body length of $14mm$ the maximal possible angular displacement is $\Delta\theta_{max} = 0.28rad$, thus the maximal produced torque by the SEE τ_{SEE} is:

$$\tau_{SEE} = \Delta\theta_{max} \cdot k_{SEE} = 1.1Nm \tag{2.2}$$

This maximal torque is very low to assist the movement of the elbow. This is why the experiments of Chapter 6 are done with a downward pendulum instead of a upward pendulum in order to reduce the total torque. Subsection 2.3.4 will discuss some possible future improvements.

2.2. Torque limitation

Given that the hardware design of F. A. Delaloye [1], which for the present is used to perform the experiments with people, has no mechanical torque limiter, the idea of adding one suggested itself. The reason why the torque has to be limited is because the motor is able to produce a stall torque M_{stall} of $2500mNm$ (see data sheet [5]) which means that after the reduction r of 126:1 of the planetary gearbox (see [6]) the human's elbow could receive a maximal torque $M_{elbow,max}$ of

$$M_{elbow,max} = r \cdot M_{stall} \cdot \eta = 236Nm \tag{2.3}$$

with $\eta = 75\%$ being the efficiency of the gearbox. To explain the order of magnitude of the maximal torque applied on the elbow, an equivalent force on the hand vertical to

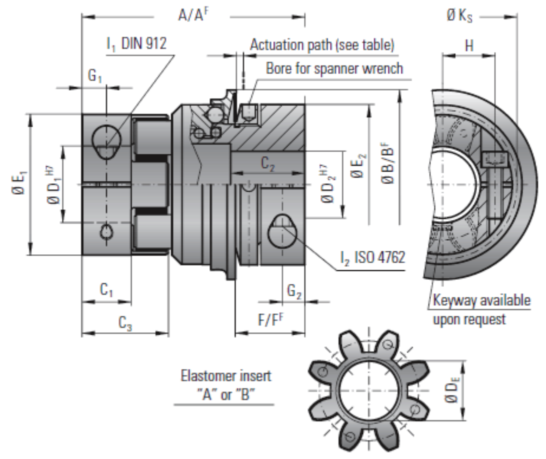
the forearm can be calculated. For a forearm length of $30cm$ the result is $788N$ which corresponds to $80kg$ approximatively. This shows that having no torque limitation is inadmissible and dangerous.
But as there was not enough place to integrate a mechanical torque limiter into the existing design, we decided to design a new platform described in Section 2.3 and to implement only a torque limitation by limiting the motor current for this setup (see Section 4.2).

2.3. Design of a new hardware setup

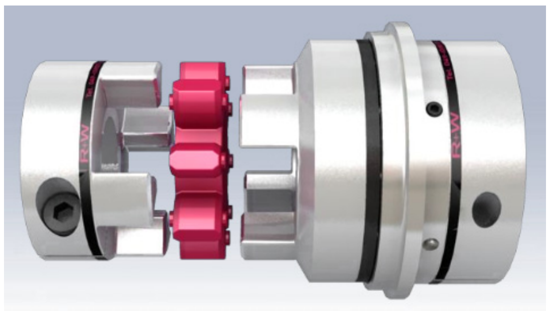
As a completely new device had to be built, the design of the SEE was put into question as well. Section 2.1 shows that the existing design of the SEE is quite limited and not very flexible. Therefore, the new setup should permit different designs of SEEs.

2.3.1. Torque limiter

The torque limiter ES2 (ES2/20/A/F/12/12/15/8-20) ([7], pp. 16-17) from R+W was chosen (see drawing in Figure 2.3a). This torque limiter provides full disengagement in case of overload and has to be re-engaged manually. In addition it possesses an elastomer insert (Figure 2.3b) which should damp vibrations and compensate for misalignments without backlash. The maximal permitted torque can be adjusted. In our case this will be set to $15Nm$ approximatively.



(a) Drawing of the torque limiter ES2.

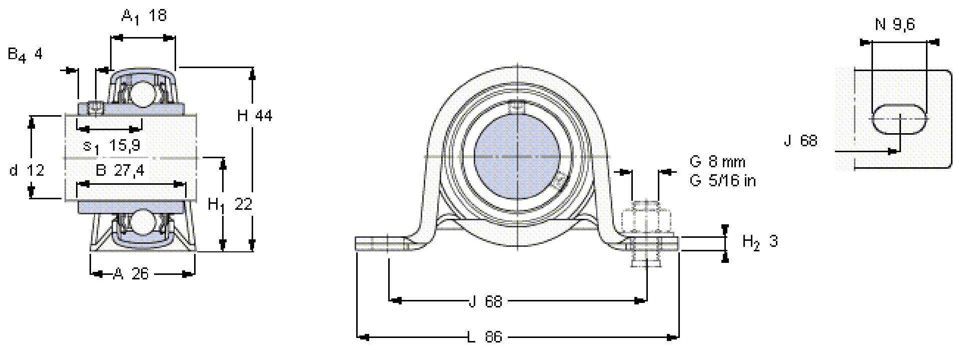


(b) Elastomer insert.

Figure 2.3.: Torque limiter ES2. Source R+W.

2.3.2. Bearings

In order to have a much simpler design new SKF Y-bearing plummer block units (P 40, YAR 203/12-2F) (see Figure 2.4) where bought instead of machining complicated parts. The advantage of these bearings is that while mounting them, the orientation of the shaft can be changed for some degrees simplifying the problem of a global mechanical alignment of two coaxial bearings.



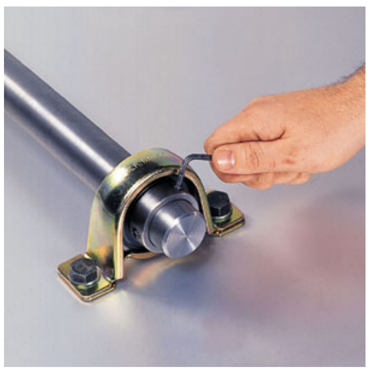
Grub screw	M 6×0.75
Recommended tightening torque [Nm]	4
Hexagonal key size [mm]	3

Figure 2.4.: Drawing of the Y-bearing plummer block unit: P 40, YAR 203/12-2F. Source SKF.

The drawback of simplifying the mechanical parts is the a little bit more delicate mounting as the bearings have to be correctly aligned by mounting them. Figure 2.5 shows the mounting steps for one bearing. As in our case two bearings are mounted with one shaft, the four screws fixing the two bearings have to be tightened up by very little incremental steps on the same time, while checking all the time if the shaft spins well.

(a) Step 1: Lay the shaft with the bearing on the housing base and place the housing cap over the bearing.

(b) Step 2: Align the unit and secure it with the attachment bolts. It is recommended that washers and spring rings be used to secure the nuts of the screws.



(c) Step 3: Axially position the shaft in the bearing and, if possible, turn it a few times. Fully tighten the grub screws in the bearing inner ring extension.

Figure 2.5.: Mounting of the Y-bearing plummer block unit. Source SKF.

To adjust the height of the shaft's position, the bearings are mounted on very simple aluminum cuboids having two clearance holes for the screws. In case the shaft should be at another height, one can simply remachine these simple blocks, instead of remachining a hole new bearing as it would have been in the case of the previous design in [1].

2.3.3. Ground plate

The ground plate is designed big enough to avoid having parts (e.g. the motor) beyond the plate's border. In addition, several female threads were created to allow numerous positions for the bearings. A total of six different configurations are possible, according to the individual space requirements (see Figure 2.6).

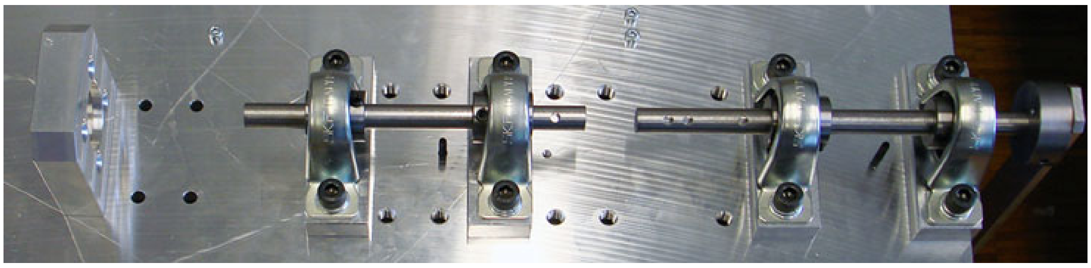


Figure 2.6.: Ground plate with one of the six configurations.

2.3.4. SEE

Unfortunately the actual design of the SEE impedes a large choice of springs. Neither at Durovis nor at other manufacturers compatible springs with a higher spring rate could be found. Therefore there are three possibilities to obtain a higher torque:

- Put several springs in parallel
- Order customized traction springs
- Change the design of the SEE

With the new device there is more space for a new SEE design, which means that it would be an easy task to put several springs in parallel. But in order to obtain around $6Nm$, 36 traction springs would be necessary. An small problem would pose the possibility of the spires of two adjacent springs getting into each other. A possible solution would be to make notches on the pins which hold the springs. This would prevent the springs from moving laterally. An other way would be to put small disks on the pins between the springs. If the diameter of these disks is chosen in corresondance to the body length of the springs, the disks could prevent the spring from being compressed and being deformated in an irreversible manner.

The second idea, to order customized traction springs with a higher spring rate and compatible lengths and deflections, sounds initially very promising, but after trying to compute new springs with the Durovis Spring Computation software (see Appendix B.1), one realizes that a SEE of this size, with a very high equivalent spring rate and a large deviation angle (e.g. 20°) is very hard to compute and nearly mechanically impossible. For a small deviation angle and a very high k_{SEE} it would be easier to find a good solution, but the drawback is that small angular perturbances (from the motor for example), would create very high torques on the elbow. A second problem of this SEE design is the fact that the springs have to be mounted under traction. Already with the actual springs (08/1/1) it requires some manual ability and time. In case of stiffer springs it would be nearly impossible to mount them. Therefore a different mounting system or some pluggable levers should be designed. Note that customized springs would cost each around 10 to 20 CHF at Durovis.

An other possibility would be the use of torsion springs. At the first view, one can observe that it would be much easier to obtain higher torques, even with only one torsion spring. The only drawbacks listed in [1] is the axial mounting. But as with the new design there is much more space according to the configuration of the bearing positions, it would be very easy to mount and unmount a torsion spring, as one of the two SEE disks could be retractable, as well as the shaft of the pendulum. The only steps to perform would be the unscrewing of the grub screws in the inner ring extension of the two "elbow bearings" and the grub screw of the "elbow encoder". There is one point from [1] which has to be amended a little bit, namely the bidirectionality of a torsion spring. First of all the spring rates of a torsion spring normally differ for the two directions, because the resistance against the direction of winding is higher, and second, a torsional leg spring should be only loaded in direction of the winding, since otherwise the leg would break after a short time. A solution to this problem would be the use of two torsion springs which are wound in opposed directions connected in series such that there is always only one of the springs loaded depending on the direction. This would require a third disk or bushing between the two springs. A disadvantage of this system could be a small discontinuity of torque at an angle of zero, but in case of a well dimensioned system, this would be negligible. Another very promising idea would be to design something similar as in [8], but without motorized stiffness regulator. In this construction very stiff springs could be used, which would result in a high k_{SEE}.

3. Electronics

By applying fast oscillations by hand to the pendulum, the EPOS produced always a device error while trying to follow the rotation due to the fact that the motor had to decelerate very fast. The motor works consequently as a generator and the energy flows from the back to the EPOS and the power supply resulting in an increased voltage. The EPOS Firmware Specification [9] suggests to put a capacitor (e.g. $2200\mu F$) to solve the problem. In our case one $2200\mu F$ and two $2350\mu F$ capacitors had to been added in parallel between the power lines from the power supply to the EPOS. In addition two diodes where added after the output of the power supply to prevent current going back to the power supply and shutting down the power supply. The result can be seen in Figure 3.1. Due to the large capacitors, when cutting the current, the motor continued to turn for around $5s$, which in case of an emergency could cause high damage. Therefore the emergency switch was displaced from the AC $230V$ to the power lines of the motor to stop the motor instantaneously (not shown in Figure 3.1).

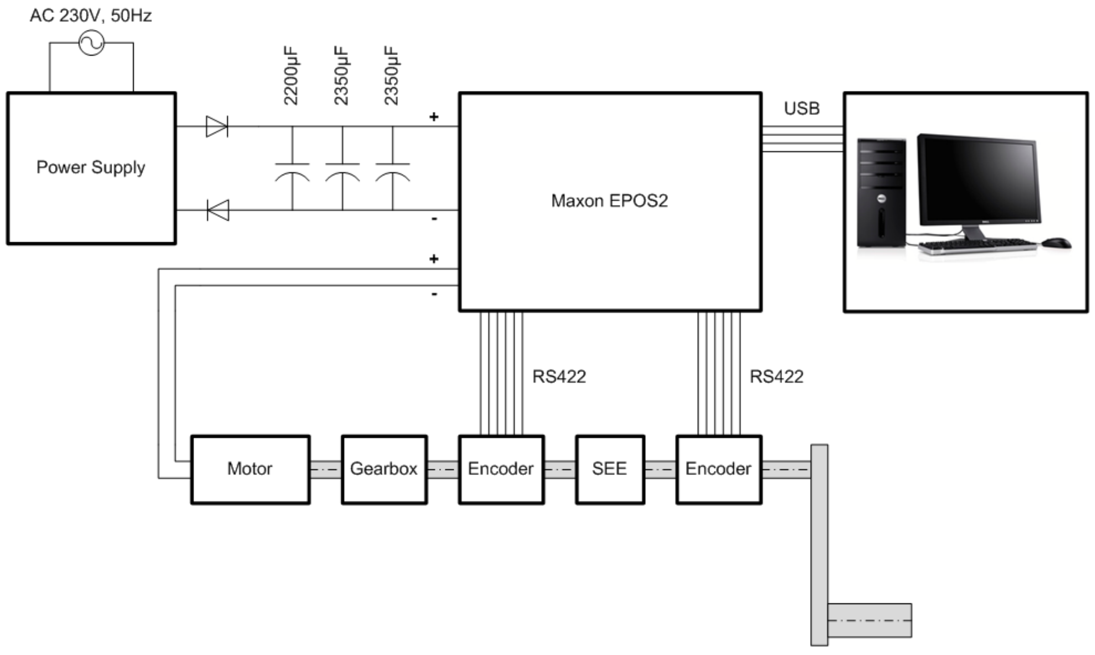


Figure 3.1.: Simplified schema including the added capacitors and diodes.

4. Low level control

4.1. PID tuning

After having connected the EPOS to the EPOS Studio software following the steps descibed in [10], the PID controller for the position and the PI for the current were dimensionned using the the "Expert Tuning". The implemented values of the PID/PI are listed in Table 4.1.

	P	I	D
Position	3655	3039	21703
Current	545	152	-

Table 4.1.: PID values for the position control and PI values for the current control.

4.2. Current limitation

The EPOS offers the possibility to limit the current of the motor, which results in limiting the torque according to the equation of a motor combining the torque M and the current I by the torque constant $k_M = 60.3mNm/A$ (see [11])

$$M = k_M \cdot I \tag{4.1}$$

An admissible torque for the human elbow M_{adm} is for our application around $15Nm$, thus one can calculate the maximal current delivered to the motor:

$$I_{max} = \frac{M_{adm}}{r \cdot \eta \cdot k_M} = 2.63A \tag{4.2}$$

So the following constants in Table 4.2 were set with the EPOS Studio:

Object name	Value
Output current limit	$2632mA$
Continuous current limit	$2632mA$
Thermal time constant	$300s$

Table 4.2.: Current limitation values.

The thermal time constant of motor winding is used to calculate the time how long the maximal output current (i.e. output current limit) is allowed for the connected motor.

Even if the datasheet gives the value of $41.6s$, it was set to $300s$. As the nominal current of the motor (i.e. maximal continuous current) is $3.12A$ and therefore higher than our maximal output current limit of $2.63A$, the thermal time constant could theoretically be infinite, which legitimates a such high value of $300s$. The output current limit and the continuous current limit had been chosen identically to avoid exceeding the torque limit at any moment.

5. High level control

The goal was to create an interface with MATLAB and Simulink as simple and intuitive as possible, such that a new user could take these functions or blocks and reuse them without long initiation. This Chapter will describe all parts in order to allow an easy use of them.

5.1. Overview

All the necessary parts, such as loading and interfacing the Maxon DLL are written in MATLAB. Therefore the user has the possibility to use only MATLAB to control the device without touching Simulink. The Simulink part is divided into two libraries, namely one composed of blocks which call the MATLAB files in order to control the device and one composed of very high level control blocks such as an adaptive oscillator, a torque estimator etc.

5.2. MATLAB

5.2.1. Maxon EPOS DLL

The Dynamic Link Library `EposCmd.dll` and its header file `Definitions.h` provided by Maxon are required to communicate with the EPOS. Since they were written for Visual C++, there appeared some variable type problems, but after making some small changes in the header file, it worked in MATLAB too, but the file `stdint.h` has to be included into the same folder. This DLL includes all possible functions listed in [12] to control the EPOS. The reason why a library in MATLAB was written (see Subsection 5.2.4), is because it is quite laborious to use the DLL directly.

5.2.2. MATLAB DLL library

The file `library_dll.m` takes care of loading and unloading the DLL into MATLAB. The function has to be called as follows:

```
library_dll(action, dll, header)
```

where the input parameters are all strings. `action` has to be either `'load_library'` or `'unload_library'`. The arguments `dll` and `header` are the two strings of the filenames inclusive their extension, in our case `'EposCmd.dll'` and `'Definitions.h'` respectively.

5.2.3. MATLAB epos class

The file `class_epos.m` defines a class in MATLAB to facilitate the handling of all properties of the EPOS, as the communication parameters, motor position, etc. Instead of giving each needed property by a parameter to a function, the whole object can be transmitted. In order to create an object named for example "MyEpos" it is sufficient to write:

```
MyEpos = class_epos;
```

Its properties can be easily read or written as for example the baudrate or the protocol stack name:

```
my_baudrate = MyEpos.Baudrate;
MyEpos.ProtocolStackName = 'EPOS2_USB';
```

5.2.4. MATLAB epos library

The file `library_epos.m` creates an easy to use MATLAB interface for the Maxon EPOS DLL. The function has to be called as follows:

```
library_epos(action, feedback, epos, arg1)
```

where the first two input parameters (i.e `action` and `feedback`) are strings and `epos` an epos object. `arg1` is an optional parameter and can be any type, but normally it is a number, for example to set the motor position. `action` can be any name of an implemented subfunction in the file `library_epos.m`. The parameter `feedback` has to be either `'feedback'` or something else (e.g. `''` or `'no feedback'`). This will determine if the called function shall display some informations in the MATLAB Command Window or not. Below there are two examples showing how to call the function. The first one has no optional argument but demands feedback, the second wants no feedback and sends the position 2100 to the EPOS.

```
library_epos('connect_device_manually', 'feedback', MyEpos);
library_epos('set_position_must', '', MyEpos, 2100);
```

Note that not all possible functions of the DLL are interfaced in this file, but only a certain choice of important functions. Thus this file could be extended if other functions would be necessary. In order to add new functions please consult [12] and look for the commented code in the file `library_dll.m` including `libfunctionsview(dll_name)`. This command will list the DLL functions with its input and output arguments in an external window. Accessorily there might be some properties to be added in `class_epos.m`.

5.2.5. MATLAB complete example

For this MATLAB example `example_matlab.m` the following files are needed (all files have to be in the same folder as the example file):

- `EposCmd.dll`

- `Definitions.h`
- `stdint.h`
- `library_dll.m`
- `library_epos.m`
- `class_epos.m`

Listing 5.1: example_matlab.m

```matlab
% create object
MyEpos = class_epos;

% load library
MyEpos.EposDll = 'EposCmd.dll';
MyEpos.EposHeader = 'Definitions.h';
library_dll('load_library', MyEpos.EposDll, MyEpos.EposHeader
   );

% create connection
MyEpos.DeviceName = 'EPOS2';
MyEpos.ProtocolStackName = 'EPOS2_USB';
MyEpos.InterfaceName = 'USB';
MyEpos.PortName = 'USB0';
MyEpos.Baudrate = 1000000;
MyEpos.Timeout = 500;
library_epos('connect_device_manually', 'feedback', MyEpos);

% get device ready
library_epos('clear_fault', 'feedback', MyEpos);

% application example
for position=0:1:1000
    library_epos('set_position_must', '', MyEpos, position);
end

% close device
library_epos('close_device', 'feedback', MyEpos);

% unload library
library_dll('unload_library', MyEpos.EposDll, MyEpos.
   EposHeader);
```

5.3. Simulink

5.3.1. Simulink epos library

The `simulink_epos_library.mdl` contains several blocks to use the EPOS in Simulink. There are two main blocks, 'Initialize EPOS' and 'Exit EPOS', which are needed to establish the communication and to close the device. The other blocks are for example to set or to get the position of the motor. As the list of blocks is not exhaustive, some more blocks can be added easily. The blocks can simply be dragged into a new Simulink Model file and be connected together.

5.3.1.1. 'Initialize EPOS' block

The 'Initialize EPOS' block does the following when starting:

1. Create an epos class object 'MyEpos'
2. Load the Maxon DLL
3. Establish the connection
4. Get the device ready (clear faults)
5. Set the device into Disable State
6. Set the maximal acceleration

To actually enable or control the EPOS other blocks are necessary. An example of a block which does further initialization is described in 5.3.2.1.
A double click on the block opens the dialog to change the block parameters. The default values are shown in brakets.

EPOS DLL tab:

- EPOS DLL (`'EposCmd.dll'`)
- EPOS Header File (`'Definitions.h'`)

Device Connection tab:

- Connection Method (`Manual`)
- Device Name (`'EPOS2'`)
- Protocol Stack Name (`'EPOS2_USB'`)
- Interface Name (`'USB'`)
- Port Name (`'USB0'`)

- Baudrate (`1000000`)
- Timeout (`500`)

System Properties tab:

- Maximal Acceleration (`25000`)

Behind this block is the Level-2 MATLAB S-Function `simulink_epos_init.m` where all the low level properties and functions of the block are defined (except the block mask). The actions listed above are all in the `Start(block)` callback of this S-Function, therefore they are only executed when the Simlulink model (i.e. "simulation") is started. Even if this block has an input and an output connection available, there is no data sent through. As this block has to be at the very beginning of the model its input is normally not connected. The successive blocks are connected to its output.
Note that the connection to the EPOS can be configured through a 'Open Device' dialog, which can be chosen in the dialog of the block parameters. For this feature the file `simulink_epos_init_callback.m` was necessary to disable de text inputs for the connection in case the 'Open Dialog' was selected.
Some tests showed that choosing a Timeout of $50ms$ can pose less problems for the 'Real Time' block described in Subsubsection 5.3.1.5 (see for more details on the timeout [9, 13]).
A further elegant step would be to add the motor torque limitation implemeted by the current limitation shown in Section 4.2 in the System Properties tab, such that the user could put the desired maximal torque, the torque constant, the reduction of the gearbox and it's efficiency in. The 'Initialize EPOS' block could then calculate the maximal current using Equation (4.2).

5.3.1.2. 'Exit EPOS' block

The 'Exit EPOS' block does the following when terminating:

1. Close the device (i.e. communication)
2. Unload the Maxon DLL

As in the case of the 'Initialize EPOS' block, the 'Exit EPOS' block is implemented by a Level-2 MATLAB S-Function `simulink_epos_exit.m`. Its actions are all implemented in the `Terminate(block)` callback which is only executed when stopping the "simulation" in Simulink. It has to be placed at the very end of the model in order to be the last executed block. Contrary to the previous block, normally only the input is connected. There is on data transmitted on the connections.

5.3.1.3. 'Get Position' block

The 'Get Position' block reads the position of an encoder (units: $20000[encoder] \mathrel{\widehat{=}} 2\pi$) and sends the value via its output connection. The block takes no input value, but its input has still to be connected to define the program flow.
A double click on the block opens the dialog to change the block parameters. The default values are shown in brakets.

- Read EPOS Position Value (`Motor Encoder`)

The only parameter is the encoder one desires to read. It can be either the encoder on the motor side or the `Master Encoder` on the elbow side. This block is implemented as the Level-2 MATLAB S-Function `simulink_epos_get_position.m`, where this time the actions are in the `Outputs(block)` callback which is executed on every cycle of the model loop.

5.3.1.4. 'Set Position' block

The 'Set Position' block `simulink_epos_set_position.m` is exactly the same as the 'Get Position' block with the difference that it takes a value as input (i.e. the demanded position of the motor) and has no output data.
A double click on the block opens the dialog to change the block parameters. The default values are shown in brakets.

- Add Master Encoder offset to position (`Off`)

This parameter defines if the offset between the master encoder (i.e. the elbow) and the motor encoder shall be added to the demanded position. this offset will be defined in 5.3.2.1 within the 'Initialize REHAB' block.

5.3.1.5. 'Real Time' block

The 'Real Time' block does actually not communicate with the EPOS, but can be very useful if the control algorithm need a constant sample time or a clock (i.e. real time). The problem with Simulink is, apart from the fact that Windows is no Real Time OS, that its simulation run much faster than real time and don't have a constant sample time. There are ways to use Simulink for some external hardware with real time, as for example with the Simulink Real-Time Workshop, or the Simulink Real-Time Windows Target in case it has to be run on the comupter itself. The problem however, is that, because Simulink has to compile the code into C code, not every arbitrary MATLAB code is accepted. Therefore it was not possible to load the Maxon DLL. There are some solutions, so called "soft real time" blocks, which are Simulink blocks placed into a Simulink model slowing down the "simulation" speed to real time. On the MathWorks site (`http://www.mathworks.com/matlabcentral/fileexchange/`) four such blocks can be found, but every block has its drawback:

- **Simulink Real Time Execution**, [14]
  Problem: Minimal time resolution of $15.6ms$

- **Real-Time Blockset7.1 for Simulink**, [15]
  Problem: Gets unstable after a certain time
- **RTsyncBlockset**, [16]
  Problem: Time steps varying from 10 to 20ms
- **Real-Time Pacer for Simulink**, [17]
  Problem: Differences on the order of 10 to 30ms

The most promising block was the first one (see [14]), due to its constant sample time. But there was invincible minimum sampling time limit of 15.6ms due to a unknow reason. In case the process would take more time (e.g. 20ms), the sampling time would have taken 31.2ms. As our goal was to reach a sample time of 10ms an other solution had to be found: our own soft-'Real Time' block.
On every cycle the block checks the duration of the cycle in a `while` loop until reaching the defined sample time. This method can consume around 15% of the CPU on an Intel i5, but the big advantage is its precision and resolution, which are beyond comparison to the four previously mentionned blocks. The idea to use `pause` instead to avoid CPU load, does not perform well, as seen in [17].
A double click on the block opens the dialog to change the block parameters. The default values are shown in brakets.

- Sample time [s] (`0.01`)

This 'Real Time' block is normally placed at the beginning of the Simulink model, therefore its input is not connected. It has three outputs: the first one is only for the program flow (the consecutive block is connected there), the second output `dt` is the actual duration of the cycle and the third output `t` is the elapsed time in seconds from the start of the "simulation" on. The two last outputs can of course be used for other functions or blocks. In addition, the block prints the elapsed time on the MATLAB Command Window, so that the user monitoring an experiment knows about the time. In case the cycle duration should take exceptionally long (i.e. twice the sampling time), `Problem` is printed on the Command Window. It can occur sometimes, that this event happens very often. In this case it is recommended to close MATLAB completely and to restart it again. Due to the fact that Windows is no Real Time OS, there will always be some problems. Therefore if the device should be commercialized or if the device should need a very precise time management, a microcontroller or a Real Time OS should definitively be considered.
The 'Real Time' block contains a Simulink Subsystem shown in Figure 5.1 which contains a Level-2 MATLAB S-Function `simulink_real_time.m` performing the wait in the `Outputs(block)` callback.

5.3.2. Simulink rehab library

The `simulink_rehab_library.mdl` includes some very high level blocks as which do not actually touch the EPOS, except from one block ('Initialize REHAB' block) which

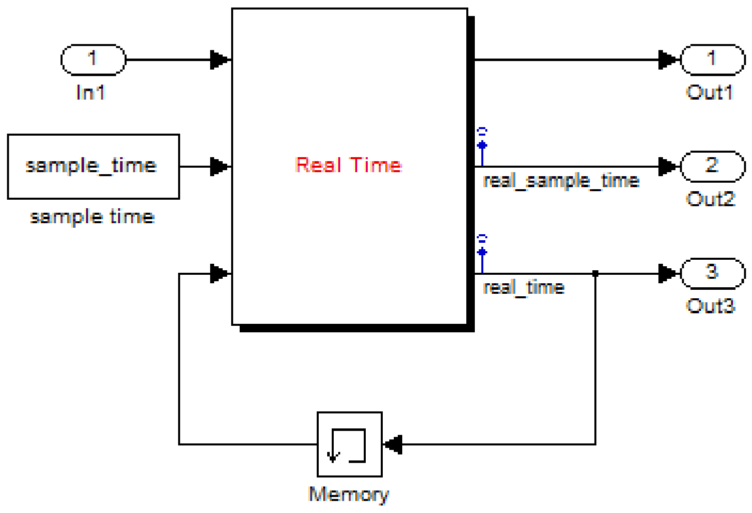


Figure 5.1.: Subsystem of the 'Real Time' block.

finalizes the initialization and takes care of the termination. The blocks can simply be dragged into a Simulink Model file and be connected together.

5.3.2.1. 'Initialize REHAB' block

The 'Initialize REHAB' block is a block for further case specific initialization and termination which has to be placed between the 'Initialize EPOS' block and the 'Exit EPOS' block. The 'Initialize REHAB' block does the following when starting:

1. Set the device to Master Encoder Mode
2. Get the offset between the master encoder and the motor encoder
3. Set the device to Position Mode
4. Set the device into Enable State

and the following when terminating:

1. Set the device to Master Encoder Mode

The 'Initialize REHAB' block, which is implemented by a Level-2 MATLAB S-Function `simulink_rehab_init.m`, measures the offset between the encoders, because this has to be taken into account later on to avoid a prompt movement of the motor at the beginning when the device is enabled. A very important point for this application, which includes human interacting with the device, is the safety. Therefore this block assures to leave the device in Master Encoder Mode after the "simulation" is stopped. In the Master Encoder Mode the motor follows simply the movement of the master encoder (i.e. elbow in our case) which means that the device does not stop abruptly but continues to move with the human until he stops his movement and releases the handgrip.
Note that for initialization the pendulum should be in the vertical position bevore starting the Simulink model and should not be touched until the running time is displayed in the MATLAB Command Window.

5.3.2.2. 'Adaptive Oscillator' block

The 'Adaptive Oscillator' block is able to estimate the kinematics of a sinusoidal movement. The adaptive oscillator is a tool developed by Righetti et al. [18, 19] and used in many applications [20, 21]. A simplified version of the modified Hopf oscillator proposed in [18] was used, by projecting this oscillator into polar coordinates and keeping the phase equation only. An augmented phase oscillator [19] can be obtained:

$$\dot{\phi}(t) = \omega(t) + \nu F(t) \cos \phi(t), \tag{5.1}$$

where $\omega(t)$ is the intrinsic frequency of the oscillator, and ν the learning parameter determining the speed of phase synchronization to the periodic input signal $F(t)$. In order to learn the frequency of the input $F(t)$, instead of doing mere synchronization only, the oscillator frequency was turned into a new state variable, integrating the phase update:

$$\dot{\omega}(t) = \nu F(t) \cos \phi(t). \tag{5.2}$$

From (5.2), it can be established that the integrator argument sums up to zero over one period (i.e. ω converges) if the frequency ω is equal to the frequency of the input signal. The adaptive oscillator input $F(t)$ was the difference between the elbow angular position $\theta(t)$ (measured by the device encoder) and the learned (i.e. estimated) position of the elbow $\hat{\theta}(t)$: $F(t) = \theta(t) - \hat{\theta}(t)$, this estimated position being the oscillator output plus the offset $\alpha_0(t)$:

$$\hat{\theta}(t) = \alpha_0 + \alpha_1 \sin \phi(t) \tag{5.3}$$

The offset $\alpha_0(t)$ and the amplitude $\alpha_1(t)$ were finally learned by the following two integrators:

$$\dot{\alpha}_0(t) = \eta F(t) \qquad \dot{\alpha}_1(t) = \eta F(t) \sin \phi(t) \tag{5.4}$$

where η is the integrator gain.
Assuming the elbow movement to be (quasi-)sinusoidal, the same adaptive oscillator can further provide an estimate of the elbow velocity and the acceleration:

$$\hat{\dot{\theta}}(t) = \alpha_1 \omega(t) \cos \phi(t) \tag{5.5}$$

$$\hat{\ddot{\theta}}(t) = -\alpha_1(t) \omega(t)^2 \sin \phi(t) \tag{5.6}$$

As in [21], the values $\nu = 20$ and $\eta = 5$ for the learning parameter and the integrator gain, respectively, were used for the experiments in Chapter 6. Good behavior using these parameters was confirmed during pilot tests.
A double click on the block opens the dialog to change the block parameters. The default values are shown in brakets.

- Learning parameter (20)

- Integrator gain (5)

Figure 5.2 shows the Simulink Subsystem of the 'Adaptive Oscillator' block. The adaptive oscillator is implemented in an Embedded MATLAB Function. The unit delay blocks are needed to keep the values of the last cycle. The output `POstate` is an array with the four values (1) offset α_0, (2) omega ω, (3) phi ϕ and (4) amplitude α_1.

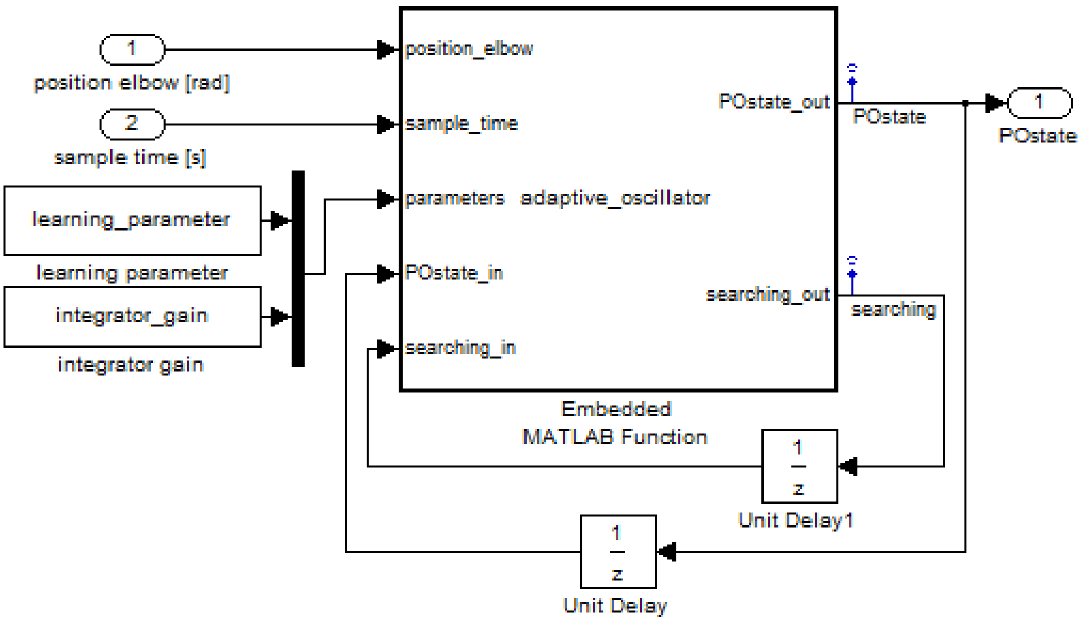


Figure 5.2.: Subsystem of the 'Adaptive Oscillator' block.

5.3.2.3. 'Torque Estimator' block

The 'Torque Estimator' block receives as input the estimation by the adaptive oscillator of the movement's kinematics and uses them to predict the needed torque by using an inverse dynamical model.
Assuming that the dynamics of the elbow (index $_h$ for "human") in parallel to the device (index $_d$ for "device") are governed by the pendulum-like differential equation for the angular position $\theta(t)$, it can be written:

$$(I_h + I_d)\ddot{\theta}(t) + (b_h + b_d)\dot{\theta}(t) + (m_h g l_h + m_d g l_d)\sin\theta(t) = \tau_h(t) + \tau_d(t) = \tau_{tot}(t) \quad (5.7)$$

where I is the inertia, m the mass, l the equivalent length, b the damping coefficient and τ the torque. The total torque $\tau_{tot}(t)$ is composed by the torque provided by the human $\tau_h(t)$ plus the torque provided by the assistive device $\tau_d(t)$. The control algorithm was based on an estimation of the total torque $\hat{\tau}_{tot}(t)$ by plugging Equations (5.3), (5.5), and (5.6) of the adaptive oscillator into (5.7), to obtain an inverse model:

$$\hat{\tau}_{tot}(t) = (I_h + I_d)\hat{\ddot{\theta}}(t) + (b_h + b_d)\hat{\dot{\theta}}(t) + (m_h g l_h + m_d g l_d)\sin\hat{\theta}(t) \quad (5.8)$$

Finally, the device provided a fraction of this estimated torque to the human in order to assist the movement:

$$\tau_d(t) = \kappa \hat{\tau}_{tot}(t) \tag{5.9}$$

where κ is the level of assistance. In theory, $\kappa = 1$ is the upper limit, corresponding to the situation where the participant should perform the movement without providing any torque. However, preliminary tests revealed that it was not possible to go over $\kappa = 1/3$ with our device, due to torque saturations, model approximations, and stability issues. In this configuration, the device should provide about one third of the total torque.
The forearm mass m_h, equivalent length l_h, and inertia I_h were individually estimated for each participant, using standard tables from [22]:

$$m_h = 0.022 M_{body} \ [kg] \tag{5.10}$$

$$l_h = 0.682 L_{forearm} \ [m] \tag{5.11}$$

$$I_h = m_h (0.827 L_{forearm})^2 \ [Nms^2/rad] \tag{5.12}$$

where M_{body} and $L_{forearm}$ denote the total body weight and the total forearm length, respectively. In order to calculate the damping coefficients b_h and b_d, respectively, the damping ratios ζ have to be known. The damping ratio of the elbow for similar movements around the upward vertical position is documented in the literature [23] and is approximatively $\zeta = 0.2$.

$$b = 2 I \omega_0 \zeta \ [Nms/rad] \tag{5.13}$$

where ω_0 is the undamped angular frequency:

$$\omega_0 = \sqrt{\frac{mgl}{I}} \ [rad/s] \tag{5.14}$$

A double click on the block opens the dialog to change the block parameters. The default values are shown in brakets.

Participant tab:

- Body mass [kg] (`65`)
- Forearm length [m] (`0.3`)
- Damping ratio [-] (`0.2`)

Device tab:

- Pendulum mass [kg] (`0.391`)
- Pendulum equivalent length [m] (`0.3296`)

- Pendulum damping ratio [-] (`0.1719`)

The default values of the device a only very approximative estimations and should be identified more precisely as a future work. Of course, the values in the Participant tab have to be adjusted individually.
This block is, as many others, a small Subsystem containing a Embedded MATLAB Function. There is nothing particular to mention of the implementation.

5.3.2.4. 'Position Estimator' block

The 'Postion Estimator' block converts the assistive torque of the 'Torque Estimator' block and converts into a difference of position betweeen the motor and the elbow, knowing the elbow position and the SEE equivalent spring rate. This difference is added to the elbow position and sent through the output of the block.
A double click on the block opens the dialog to change the block parameters. The default values are shown in brakets.

- Spring rate of SEE [Nm/rad] (`3.8`)
- Maximum SEE deviation [rad] (`0.25`)

The maximal permitted angular deviation of the SEE depends on the body length of the springs, as well as on the size of the disks. The difference of the position of the motor and the elbow should never exceed this value, because if it did, the human would perceive the direct torque of the motor after the reduction. A minor side effect could be an irreversible deformation of the springs. Thus, the position of the motor is being "saturated" in such a case and so the torque as well. As the system doesn't react immediately, precaution should be taken to put in a slightly smaller number as the real value.
As the previous block, this block is a small Subsystem containing a Embedded MATLAB Function.

5.3.2.5. Unit converter blocks

These blocks are very simple Simulink Function Blocks which convert the encoder pulse-counts to radian ('[encoder] to [rad]' block) and vice versa ('[rad] to [encoder]' block) according to the equation:

$$20000[encoder] \mathrel{\widehat{=}} 2\pi \tag{5.15}$$

The value 20000 comes from the fact that the encoder is an incremental quadrature encoder with 5000 pulses (i.e. $5000 \cdot 4 = 20000$). For more details see datasheet [24]. These blocks will be found often at the output of a 'Get Position' block or at the input of a 'Set Position' block.

5.3.2.6. 'Protocol' block

The 'Protocol' block is contained in `simulink_rehab_library.mdl` too, even if it is very case specific. The protocol defines the assistance level as a function of the time. As this block is implemented in an Embedded Matlab Function and takes as input the time, it could produce any profile of assistance. The way the 'Protocol' block in this library works, is by defining a profile of several successive trapezoids, where the start time, end time, slope time and height (i.e. assistance level) can be defined for each trapezoid independently (see en example in Figure 5.3). Afterwards, a succession of `if` conditions to check the time in respect to the trapezoids is programmed. Then it applies the correct assistance level to its first output connection, corresponding to the profile of the trapezoid. Its second output `nb` is used in case one would like to know which of trapezoids is active, therefore its output is the number of order of the trapezoid.

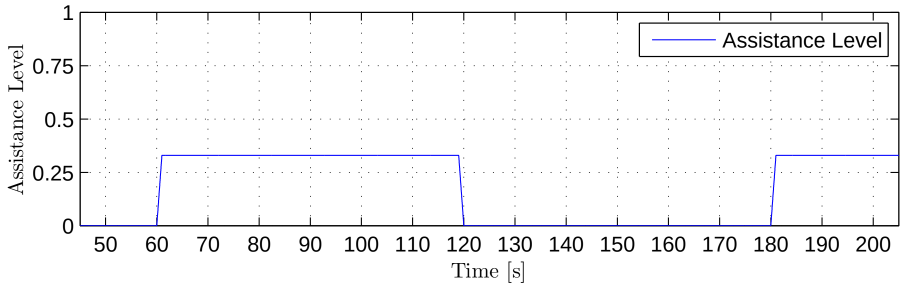


Figure 5.3.: Trapezoid of an assistance protocol.

This kind of implementation looks kind of brute, but as this 'Protocol' block could take any form, it is not worth to implement something elegant. A very elegant solution would be a graphical tool, in which the user could draw its profile by pieces of lines, which would snap in an intelligent manner to the horizontal for example, and where the user could drag and drop the vertices of the profile to adjust it.
Exactly as the other block, this 'Protocol' block can be dragged into any Simulink model, but the problem is that it can not be edited, without editing the source block in the library. Therefore, if the block has to be changed, one can simply right click on the dragged block, go to 'Link Options' and 'Disable Link' and repeat it again with 'Link Options' and 'Break Link', to delete the link between the dragged block and the library permanently in order to modify the block.

5.3.2.7. 'Torque Estimator with Error' block

For the experiments in Chapter 6 we used a modified version of the 'Torque Estimator' block of Subsubsection 5.3.2.3: the 'Torque Estimator with Error' block.
In order to see the influence of a wrong estimation of the parameters I_h, b_h and l_h on the movement's frequency this block introduces some errors. The error on the mass has been omitted, because it would show exactly the same effect as the error on the length. First of all some simplifications and assumptions were introduced:

- The device was assumed to be transparent to the user, such that its inertia, damping coefficient, and mass were not taken into account in (5.7): $I_d = 0$, $b_d = 0$ and $m_d = 0$.
- The estimated kinematic variables ($\hat{\theta}$, $\hat{\dot{\theta}}$, $\hat{\ddot{\theta}}$) were assumed to be equal to the actual ones.
- As the movement is only of a small angular amplitude around the vertical position, the approximation of small angles $\sin\theta \approx \theta$ is being made.

It results in the equation (5.16) for the device's torque where the indices $_h$ and $_d$ are being omitted for the inertia, mass, length and damping coefficient for sake of simplicity.

$$\tau_d = \kappa \left(\hat{I}\ddot{\theta} + \hat{b}\dot{\theta} + \hat{m}g\hat{l}\theta \right) \tag{5.16}$$

$\hat{I}$, $\hat{b}$ and $\hat{l}$ are the wrongly estimated parameters of the human as following:

$$\hat{I} = (1 + e_I)\, I \qquad \hat{b} = (1 + e_b)\, b \qquad \hat{l} = (1 + e_l)\, l \tag{5.17}$$

where $e_{I/b/l}$ are the errors which will be created by the 'Error Protocol' of Subsubsection 5.3.2.8.
As the device's parameters are not taken into account, the Device tab of the block parameters has been removed.
A double click on the block opens the dialog to change the block parameters. The default values are shown in brakets.

Participant tab:

- Body mass [kg] (`65`)
- Forearm length [m] (`0.3`)
- Damping ratio [-] (`0.2`)

5.3.2.8. 'Error Protocol' block

The 'Error Protocol' block used for the experiments of Chapter 6 creates percentaged errors on the length, the inertia and on the damping coefficient (i.e. friction), corresponding to a certain matrix defined in the MATLAB Workspace. The block takes as input the number `nb` of the trapezoid created by the 'Protocol' block. By this number the 'Error Protocol' knows when an assistance is provided and can introduce an error accordingly. This error will then be introduced into the 'Torque Estimator with Error' described in Subsubsection 5.3.2.7 by connecting the ouputs with the corresponding inputs.
A double click on the block opens the dialog to change the block parameters. The default values are shown in brakets.

- Error Protocol Matrix (`error_protocol_matrix`)

- Length Error Amplitude [%] (`50`)
- Inertia Error Amplitude [%] (`50`)
- Friction Error Amplitude [%] (`50`)

The error protocol matrix can be any matrix, but needs 3 rows for the parameters length, inertia and damping coefficient and exactly as many columns as trapezoids defined in the 'Protocol' block. For the experiments shown in Chapter 6 two error protocols were needed. One protocol without errors and one protocol with a positive and negative error on each parameter. The errors of the latter protocol had to be distributed randomly in time, therefore a more complex structure of the block had to be created.
The file `create_training_protocol.m` creates a 3 by 7 matrix with zeros, as for the training no errors have to be introduced.

```
training_protocol_matrix =

  0   0   0   0   0   0   0
  0   0   0   0   0   0   0
  0   0   0   0   0   0   0
```

The file `create_error_protocol.m` takes as an input the number of random protocols `nb_protocols` and creates as much 3 by 7 submatrices in one three dimensional matrix. Below an example is being shown:

```
error_protocol_matrix(:,:,25) =

  0   0  -1   0   0   1   0
 -1   0   0   1   0   0   0
  0   0   0   0  -1   0   1
```

Note that if MATLAB is restarted, the `error_protocol_matrix` has to be recreated, because Simulink starts with the same random series due to a lack of random initialization. Figure 5.4 shows the Simulink Subsystem of the 'Error Protocol' block with as a core an Embedded MATLAB Function. The memory block chooses a random index of a submatrix of the error protocol matrix chosen in the block parameters dialog and keeps it for the whole "simulation". Then this submatrix of that index is loaded in the Embedded MATLAB code and updates its logic outputs according to the number of the active trapezoid. Then the errors are amplified with its individual gains in order to be sent to the 'Torque Estimator with Error'.

5.3.3. Simulink complete example

For this Simulink example `example_simulink.mdl` the following files are needed (all files have to be in the same folder as the example file):

MATLAB files:

- `EposCmd.dll`

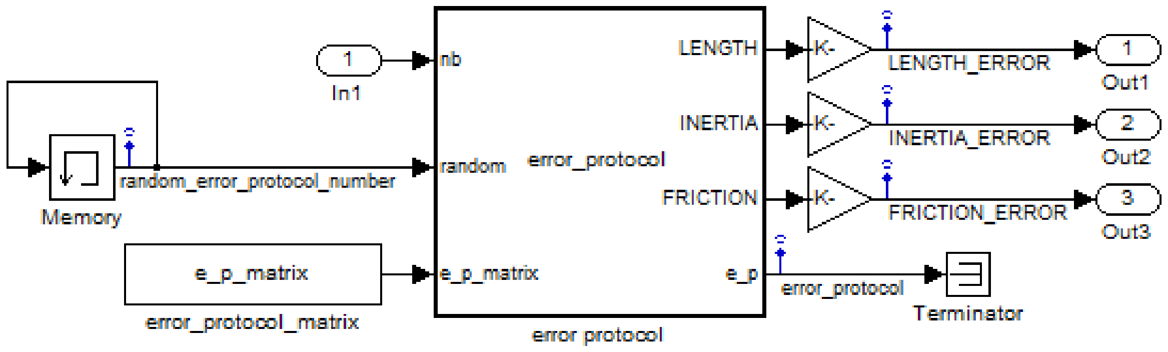


Figure 5.4.: Subsystem of the 'Error Protocol' block.

- `Definitions.h`
- `stdint.h`
- `library_dll.m`
- `library_epos.m`
- `class_epos.m`

Simulink files:

- `simulink_epos_library.mdl`
- `simulink_rehab_library.mdl`
- `simulink_epos_init.m`
- `simulink_epos_init_callback.m`
- `simulink_epos_exit.m`
- `simulink_epos_get_position.m`
- `simulink_epos_set_position.m`
- `simulink_real_time.m`
- `simulink_rehab_init.m`

The following model configurations of the Simulink example file have to be chosen:

Solver:

- Simulation time > Start time: `0`
- Simulation time > Stop time: `inf`

- Solver options > Type: `Fixed-step`
- Solver options > Solver: `discrete`
- Solver options > Fixed-step-size: `0.1`

Data Import/Export:

- Save to workspace > Time: `on`, `tout`
- Save to workspace > Signal logging: `on`, `logsout`
- Save options > Limit data points to last: `off`
- Save options > Decimation: `1`
- Save options > Format: `Array`
- Save options > Returns a single object: `off`

The Fixed-step-size of the Solver options actually has no influence at all, because it is the 'Real Time' block which dictates the time.
Figure 5.5 shows the complete Simulink model. Note that there is no classical "control loop" but two subprogram flows instead. This is the way the blocks should be used. One program flow with the initialization and termination blocks (i.e. 'Initialize EPOS', 'Initialize REHAB' and 'Exit EPOS' blocks), where the application specific block is in the middle, and the other program flow with the actual control of the device. Note that the starting block is the 'Real Time' block, because successive blocks need for example the sample time value, therefore the 'Real Time' block has to be placed ahead. All the open endings of the subprograms are equipped with a flow terminator. Once Simulink has gone through all the blocks, it will start again at the beginning. It is important to avoid closing the second subprogram flow to a loop, because Simulink would may be start with the wrong block. There is the possibility to give a certain priority to the blocks, but with an open structure it is simpler and more evident.

5.4. Signal logging

In crucial part an experiment is the collection of data for a post-hoc processing. Unfortunately it is not possible to plot the recorded data instantaneously in Simulink, because it is very time consuming. Therefore the data has to be recorded in a different way: the Simulink Signal logging.
In order to add a signal to the logging, the connection has to be right clicked. In the appearing context menu choose 'Signal Properties...'. Then the signal's name has to be chosen and the checkbox 'Log signal data' checked. Make sure that the checkbox 'Limit data points to last' is not checked. The same is valid for the checkbox of same name in the model configurations under 'Data Import/Export'. Otherwise the data array will be cut off after a certain number of cycles.

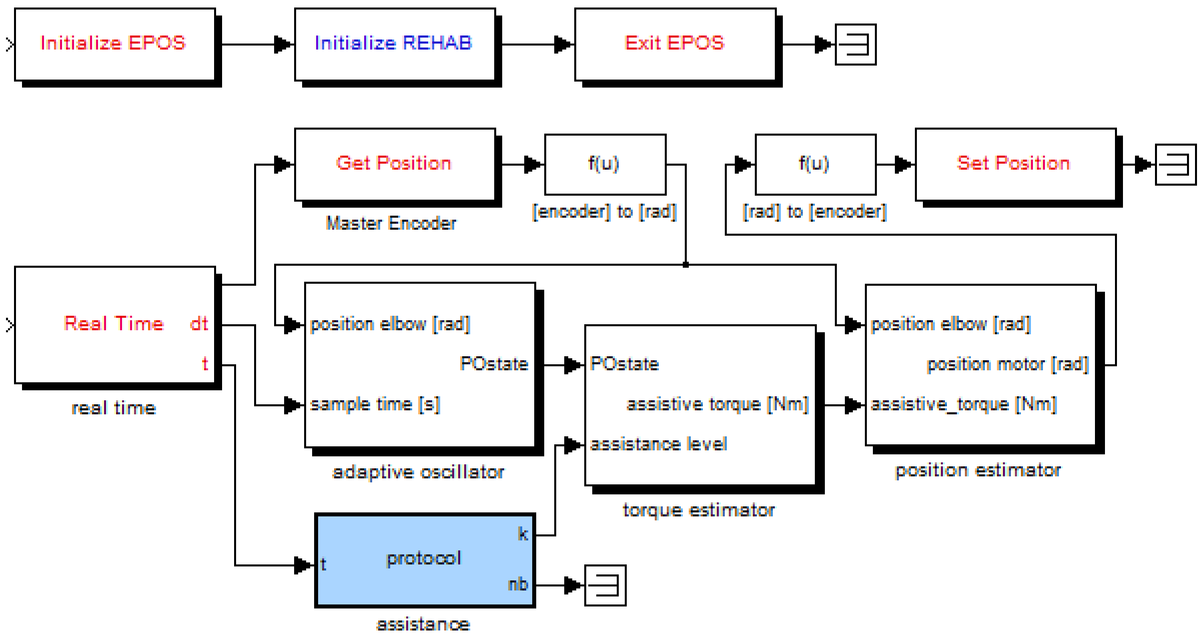
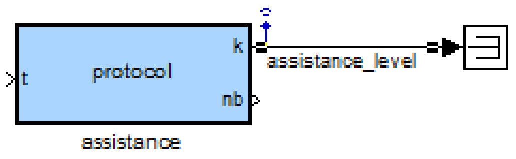
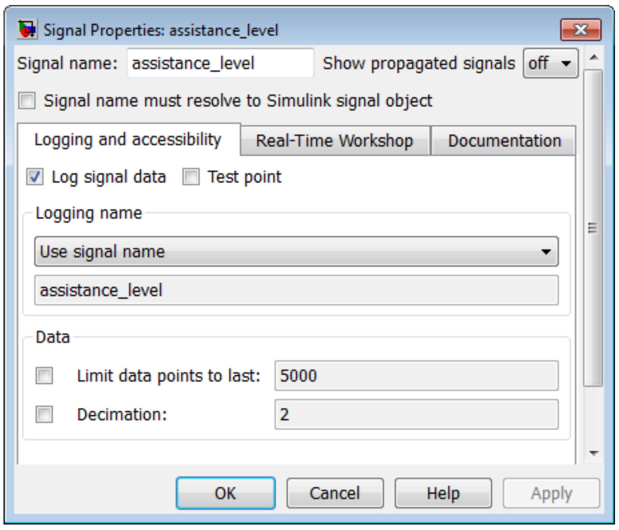


Figure 5.5.: example_simulink.mdl.



(a) Example of signal logging.



Signal Properties: assistance_level

Signal name: assistance_level   Show propagated signals: off

☐ Signal name must resolve to Simulink signal object

Logging and accessibility | Real-Time Workshop | Documentation

☑ Log signal data ☐ Test point

Logging name

Use signal name

assistance_level

Data

☐ Limit data points to last: 5000

☐ Decimation: 2

OK Cancel Help Apply

(b) Signal Properties menu.

Figure 5.6.: Signal logging in Simulink.

In case a single value as for example the body mass of the participants wants to be logged, one can check 'Limit data points to last' and enter 1.
Note that many subsystems of the previously described blocks contain already signals which are being logged, as for example in the 'Real Time' block.
After having stopped the "simulation" and being returned to MATLAB, the data is located in the variable `logsout`, which is a structure, in the MATLAB Workspace. The easiest way to find out how the data is structured inside `logsout`, is to type simply

```
>> logsout
```

in the MATALB Command Window (without the ">>"). This will show different elements as for example `assistance_level` which is of the class 'Timeseries'. To access its data, type the following:

```
>> logsout.assistance_level.Data
```

The property `.Time` of `assistance_level` is the same for all the logged data and contains the Simulink simulation time at every cycle. Note that this time has nothing to do with the real time, thus normally it will not be used.
If a signal is logged from inside a subsystem, the shown element is of the class 'Subsys-DataLogs'. In this case you can type for example for the 'Real Time' block:

```
>> logsout.('real time')
```

The two elements, `real_sample_time` and `real_time`, will be displayed. They can be accessed as the previous example:

```
>> logsout.('real time').real_sample_time.Data
```

The data is always structured as a vertical array and can be used as is in a post-hoc processing script in MATLAB. If one desires for example to plot the elbow and the motor position as a function of the time, a sample script could look like this:

Listing 5.2: example_logging.m

```
time = logsout.('real time').real_time.Data;
elbow_position = logsout.elbow_pos.Data;
motor_position = logsout.motor_pos.Data;

figure;
plot(time, elbow_position, time, motor_position);
title('Positions');
legend('elbow', 'motor');
xlabel('Time [s]');
ylabel('Position [encoder] (2 PI = 20000)');
```

6. Experiments

The aim of the following experiments was to show the influence of a bad estimation of the human's parameters (i.e. inertia, damping coefficient and equivalent forearm length) on the movements frequency. Therefore a sensitivity analysis was established to evaluate the robustness of the used method on model approximations. This Chapter corresponds to the main part of the publication "Assistance using adaptive oscillators: Sensitivity analysis on the resonance frequency" by Rinderknecht et al. for the International Conference on Rehabilitation Robotics 2011.

6.1. Simulink Model

Figure 6.1 shows the complete Simulink model used for the experiments. The corresponding files inclusive the files for post-hoc processing are listed in Appendix B.7.

6.2. Error in parameter identification: Model-based predictions

The parameters percieved by the human while being assisted are the following:

$$\tilde{I} = I - \kappa\hat{I} \qquad \tilde{b} = b - \kappa\hat{b} \qquad \tilde{ml} = ml - \kappa\hat{ml} \tag{6.1}$$

We obtain a resulting dynamical system, which corresponds to what the human perceives when assisted:

$$\tau_h = \tilde{I}\ddot{\theta} + \tilde{b}\dot{\theta} + \tilde{m}g\tilde{l}\theta \tag{6.2}$$

The system's resonance frequency can be calculated from Equation (6.2):

$$\omega_r = \omega_0\sqrt{1 - 2\zeta^2}, \tag{6.3}$$

where

$$\omega_0 = \sqrt{\frac{\tilde{m}g\tilde{l}}{\tilde{I}}} \tag{6.4}$$

is the natural frequency, i.e. the resonance frequency of the undamped dynamics, and $\zeta = \tilde{b}/(2\tilde{I}\omega_0)$ is the damping factor.
As shown in Equations (6.3) and (6.4), the natural and resonance frequencies vary in function of the system's parameters $\tilde{I}$, $\tilde{b}$, $\tilde{m}$, and $\tilde{l}$. However, if the system's parameters

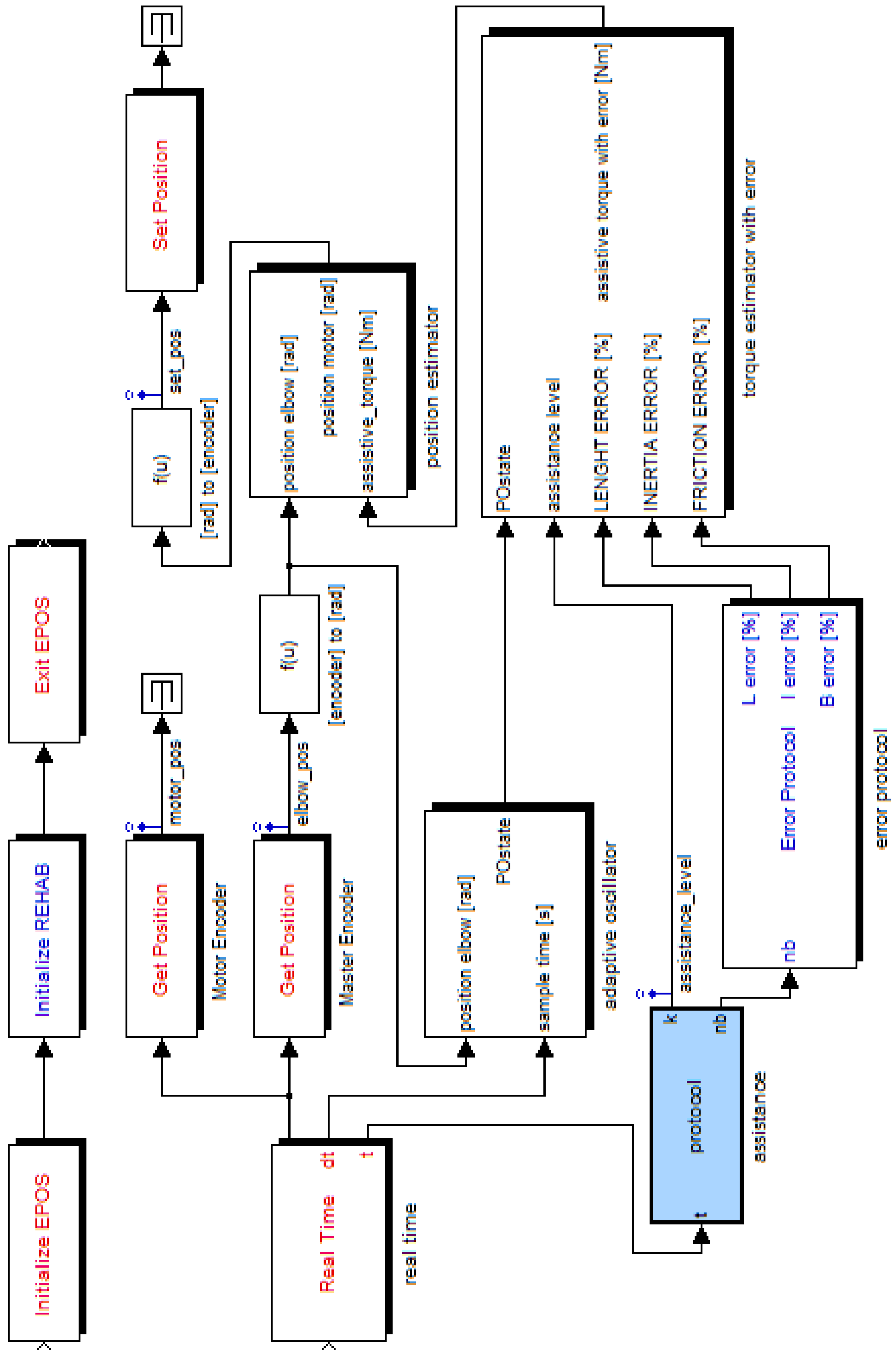


Figure 6.1.: Complete Simulink model used for the experiments.

were correctly estimated (i.e. if $\hat{I} = I$, etc. . .), the natural and resonance frequencies should not change with the level of assistance κ.
Let us now assume that the system's parameters were not correctly estimated, i.e.:

$$\hat{I} = (1 + e_I)\, I \qquad \hat{b} = (1 + e_b)\, b \qquad \hat{l} = (1 + e_l)\, l \tag{6.5}$$

where e_I, e_b, and e_l denote the errors of the corresponding variables.
Model predictions for the variation of the natural and resonance frequencies as a function of the wrongly estimated parameters are shown in Figure 6.2, using the forearm intrinsic parameters of a representative participant.

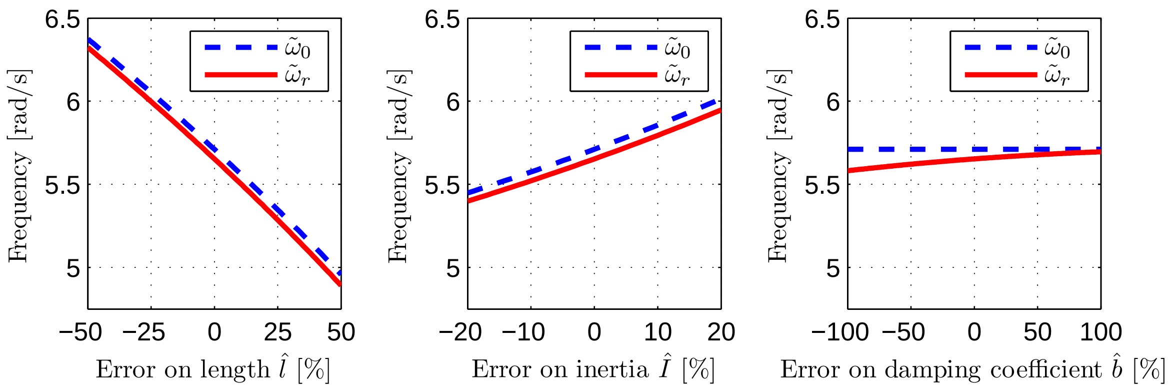


Figure 6.2.: Variation of the predicted frequencies $\tilde{\omega}_0$ (dashed blue) and $\tilde{\omega}_r$ (solid red) as a function of errors [%] on the estimated equivalent length $\hat{l}$, inertia $\hat{I}$ and damping coefficient $\hat{b}$.

Assuming that the participant performed the movement at the resonance frequency, which was therefore adapted as a function of the resulting dynamics, the results shown in Figure 6.2 lead to the following predictions:

- overestimating the equivalent length $\hat{l}$ should lead to a smaller movement frequency, and vice-versa;
- overestimating the inertia $\hat{I}$ should lead to a larger movement frequency, and vice-versa;
- overestimating the damping coefficient $\hat{b}$ should lead to a larger movement frequency, and vice-versa.

In addition, we point out that the influence of the errors on the estimation of $\hat{I}$ and $\hat{l}$ is higher than the influence of the error on the damping coefficient $\hat{b}$. This will be investigated by sensitivity analyses.

6.3. Participants

Five healthy, male participants took part to the experiment (age: 20-30years, weight: 60-90kg). The forearm mass m, equivalent length l, and inertia I were individually calculated for each participant, using the Equations (5.10), (5.11) and (5.12) and the estimated values M_{body} and $L_{forearm}$. For all participants, we used the damping ratio

$\zeta = 0.1$, i.e. approximatively half smaller than documented in the literature for similar movements around the upward vertical position [23]. This was obtained after manual tuning during preliminary tests.

6.4. Experimental protocol

Participants sat on a chair while having the right upper arm resting horizontally on a support mounted on a table and the forearm hanging down, as shown in Figure 6.3. They were asked to grab a handle attached to the assistive device, and to make cyclical flexion/extension of the elbow about the vertical downward position.

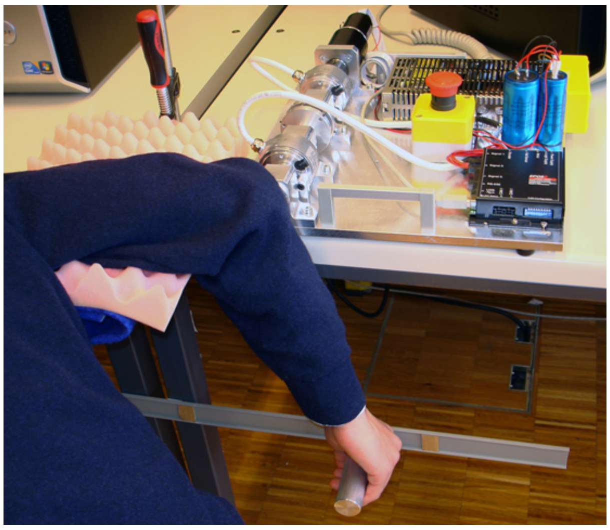


Figure 6.3.: Side view of a participant using the assistive device.

Participants wore a hearing protector to decrease potential auditory feedback from the motor noise, and had direct visual feedback about the forearm movement. The instruction was to oscillate at a constant amplitude specified by two visual markers. Importantly, the movement frequency was left free and the participants were asked to oscillate at the most comfortable frequency.

Before doing the actual data acquisition, a training sequence of seven minutes was given to the participants, in order to become familiar with the device. This sequence was made of seven trials of one minute each, alternating between no assistance ($\kappa = 0$), and movement assistance of $\kappa = 1/3$. After this training sequence, all participants reported to be relaxed with the device and the assistance. Moreover, visual inspection of the data revealed that they had all found a steady-state movement frequency. Data corresponding to the training sequence were not recorded.

After a short break, participants did a second sequence of fifteen minutes long. Again, this sequence was made up with a series of fifteen one-minute-long trials, alternating between no assistance ($\kappa = 0$), and assistance with $\kappa = 1/3$. On top of that, during the seven trials with assistance, an error in the estimation of the equivalent length e_l, inertia e_I, or damping coefficient e_b of the human forearm model was introduced. The different conditions were individually randomized for each participants, and comprised the following:

- $(e_l = -50\%, e_b = 0, e_I = 0)$, denoted $\hat{l}_-$,
- $(e_l = +50\%, e_b = 0, e_I = 0)$, denoted $\hat{l}_+$,
- $(e_l = 0, e_b = 0, e_I = -20\%)$, denoted $\hat{I}_-$,
- $(e_l = 0, e_b = 0, e_I = +20\%)$, denoted $\hat{I}_+$,
- $(e_l = 0, e_b = -100\%, e_I = 0)$, denoted $\hat{b}_-$,
- $(e_l = 0, e_b = +100\%, e_I = 0)$, denoted $\hat{b}_+$,
- and $(e_l = 0, e_b = 0, e_I = 0)$, i.e. no error, denoted $\hat{l}/\hat{I}/\hat{b}_0$.

6.5. Data processing and statistical analyses

The measured position of the elbow $\theta(t)$ was low pass filtered using a foward/backward Butterworth filter of 3rd order with a cutoff frequency of 4Hz. Moreover, the movement offset was filtered out by filtering the signal using a similar lowpass Butterworth filter, but with a cutoff frequency of 0.1Hz, and by subtracting this signal to the original one. Movement frequency was estimated by extracting the zero crossings of the filtered signal to calculate its instantaneous period. The instantaneous frequency was calculated by taking the inverse of the period and multiplying by 2π.
To compare the measured frequencies between the different conditions, statistics were performed on the data from the second half of each trial, such that only steady state performance was analyzed. To establish the significance of the different observed movement frequencies, Wilcoxon rank-sum tests were performed. This test reveals whether the median of two different populations are different.
To establish whether the observed movement frequencies complied with quantitative predictions from the model, we performed sensitivity analyses, i.e. the sensitivity of the steady-state frequency with respect to the three parameters was calculated. For the equivalent length, this was equal to:

$$S^{\omega}_{\hat{l}} = \frac{\Delta\bar{\omega}_{\hat{l}}}{\bar{\omega}_{\hat{l}_0}} \cdot \frac{\hat{l}_0}{\Delta\hat{l}}, \tag{6.6}$$

where $\Delta\bar{\omega}_{\hat{l}} = \bar{\omega}_{\hat{l}_+} - \bar{\omega}_{\hat{l}_-}$; $\bar{\omega}_{\hat{l}_-}, \bar{\omega}_{\hat{l}_0}, \bar{\omega}_{\hat{l}_+}$ denote the median of the measured frequency during the corresponding conditions; and $\Delta\hat{l}/\hat{l}_0 = 100\%$, since errors of $\pm 50\%$ were tested. Similar sensivities were computed for the inertia and the damping coefficient, $S^{\omega}_{\hat{I}}$ and $S^{\omega}_{\hat{b}}$, respectively, with $\Delta\hat{I}/\hat{I}_0 = 40\%$ and $\Delta\hat{b}/\hat{b}_0 = 200\%$.

6.6. Results

6.6.1. Movement frequency

Figure 6.4 shows both the "actual" instantaneous frequency (computed form the measured position) and the estimated frequency ω from the adaptive oscillator during the whole sequence of a representative participant. Both fit very well, as shown in the figure.

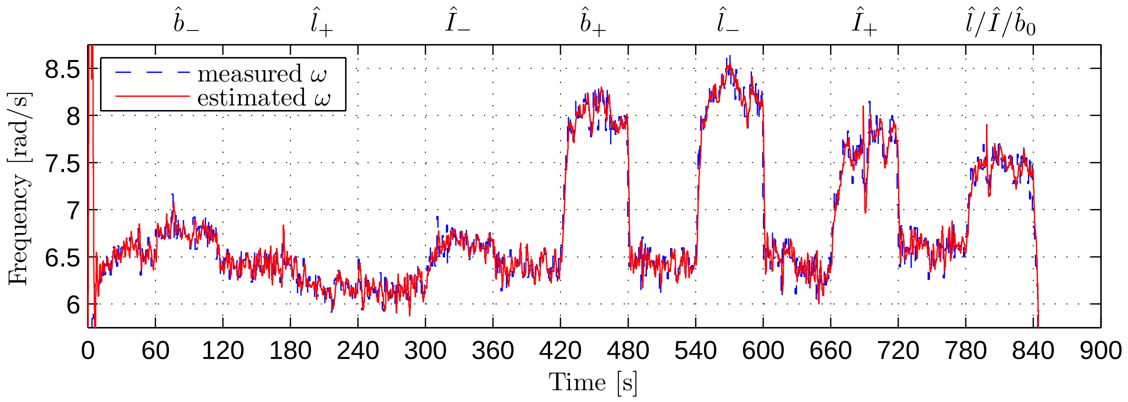


Figure 6.4.: Evolution of the measured frequency (dashed blue) and the estimated frequency by the adaptive oscillator (solid red) in time. The label on top of the graph shows the condition for the corresponding trial.

Figure 6.5 shows the steady-state frequencies of a representative participant in the different conditions. Variations are clearly visible across conditions, and, as expected from model predictions, a positive error on the length e_l induced smaller frequency, and a positive error on the inertia e_I or on the damping coefficient e_b induced larger movement frequency.

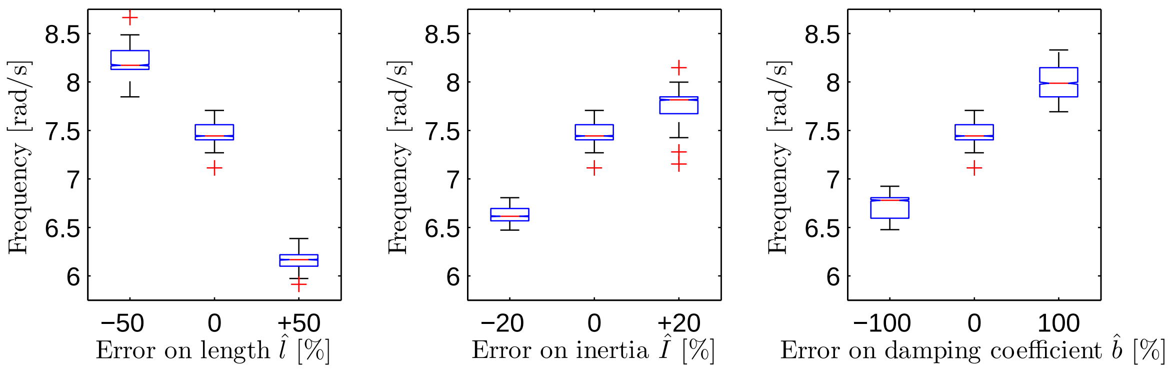


Figure 6.5.: Movement frequency for the different conditions, i.e. as a function of the error on the estimated equivalent length $\hat{l}$, inertia $\hat{I}$, and damping coefficient $\hat{b}$.

Table 6.1 shows the results for all participants, such that 9 (tests/participant) $\times$ 5 (participants) = 45 tests were performed. Wilcoxon rank-sum tests revealed that these 45 comparisons all reached significance on the population medians, with a confidence level of more than 99.9%. Regarding the predictions, 39 (87%) of these tests validaded the model-based assumptions (as shown by a ✓ in Table 6.1) and 6 (13%) revealed a trend being opposed to the corresponding model-based assumption (as shown by a ✕ in Table 6.1).

Table 6.1.: Accordance of the relationships between medians of measured frequency for two different conditions with the model-based predictions. All p-values are statistically significant ($p < 0.001$).

	$L_- > L_0$	$L_0 > L_+$	$L_- > L_+$	$I_- < I_0$	$I_0 < I_+$	$I_- < I_+$	$B_- < B_0$	$B_0 < B_+$	$B_- < B_+$
Participant 1	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$
Participant 2	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✕ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$
Participant 3	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✕ $p < 0.001$	✓ $p < 0.001$	✕ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$
Participant 4	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✕ $p < 0.001$	✕ $p < 0.001$	✕ $p < 0.001$
Participant 5	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$	✓ $p < 0.001$

6.6.2. Sensitivity analysis

The model predicted that the changes in resonance frequency has not the same sensitivity to changes in the different estimated parameters. For instance, an overestimation of $\hat{I}$ and $\hat{b}$ should both lead to larger resonance frequency (as observed for most of the participants, see Table 6.1), but at a much higher rate (higher slope) for $\hat{I}$ than $\hat{b}$. This can be observed form the slopes in Figure 6.2.

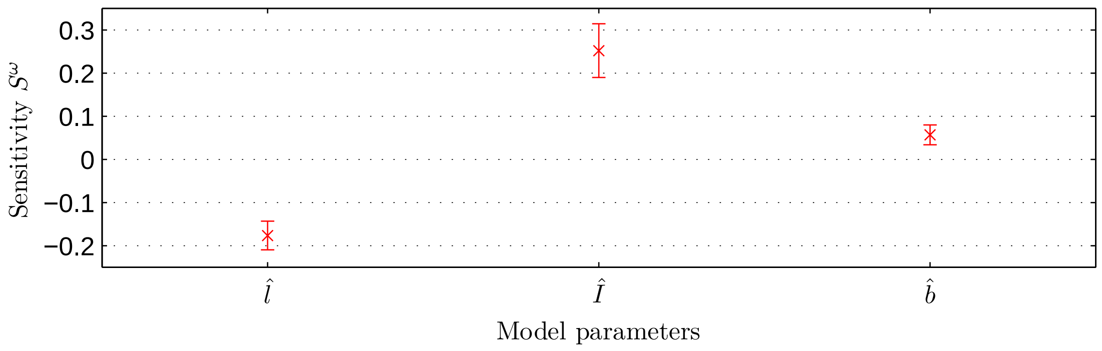


Figure 6.6.: Sensitivity of the movement frequency to variations in the estimated model parameters, namely the equivalent length $\hat{l}$, inertia $\hat{I}$, and damping coefficient $\hat{b}$. Mean ± standard error of the mean among the 5 participants.

To establish whether the observed movement frequencies complied with these quantitative predictions, sensitivity analyses were performed, as explained in Section 6.5. The results are shown in Figure 6.6 and are coherent with the model-based predictions. As expected, $S^{\omega}_{\hat{l}}$ and $S^{\omega}_{\hat{I}}$ are higher in absolute value than $S^{\omega}_{\hat{b}}$, and the sensitivity to the error in equivalent length $S^{\omega}_{\hat{l}}$ is negative, while the other two are positive.
The small sensitivity of the participants behavior to the estimate of the damping coefficient is a good point to establish the robustness of our assistive method to model approximations, since the damping coefficient b is by far the most difficult parameter to estimate.

7. Conclusion

The main part of the project was to create an intuitive and easy to use interface within Simulink, where a new user could design a control structure for the assistive device. An appropriate library with many functions and blocks has been created which could be used either with MATLAB only or together with Simulink. Unfortunately it posed more problems than expected to realize a constant sample time for the real time device control, but a workaround with the soft 'Real Time' block has been found. In terms of 'Real Time'-performance, MATLAB and Simulink can be sufficent for this kind of preliminary research on assistive devices using adaptive oscillators, but in case of commercialization of such an assistive device for the elbow, it would be very advisable to implement the whole high level control including the adaptive oscillator and the other modules on a microcontroller, or on a Real Time OS.
The other points, as for example the finalization of the electronics and the choice of springs for the existing SEE, have been fulfilled too. There has been several small attempts of a system identification of the device, with the aim of having a more accurate model, but they were all discarded, due to lack of accuracy and benefit for the experiments.
As expected, some first experiments of assisting the participant's movement by a Serial Elastic Actuator controlled by an adaptive oscillator have been carried out and the sensibility of the estimation of some model parameters on the resonance frequency have been studied. As a supplement, a pulication about these experiments and the results could be written for the International Conference on Rehabilitation Robotics 2011, taking place in Zürich.
In addition to all that, a new hardware setup including a security clutch has been created in order to fulfill the safety issues. The new device was designed to be much more flexible than the old one, such that many different SEE designs could be integrated. At the same time a significant lower manufacturing price was achieved. Futhermore a housing for the electronics, which was missing in the old setup, was designed to prevent electric shocks from exposed $230V$ cables and contacts.

8. Future work

The next steps I would propose for this project, would be to perform an iteration on the SEE design and develop an optimal and more adequate solution for the new device. An example of a completely different design which could be adapted is shown in [8]. A second step would be realizing a very accurate system identification of the new hardware setup, in order to integrate its dynamical model and its parameters into the control loop. This system identification should produce more precise results than the identification of Appendix B.8 using a simple approximated inertia, calculated using the equations of a free pendulum and the measurement of the free oscillation frequency. In my opinion these two points are very crucial for the progress of this rehabilitation project. Therefore a student with a large background in mechanics, control engineering and dynamical systems should be taken into consideration for these future steps.

9. Acknowledgements

Special thanks to Renaud Ronsse and Alessandro Crespi for the profitable discussions and their excellent supervision as well as to Auke Jan Ijspeert for enabeling this project. Thanks to all the volunteers which participated in the experiments and to Jean-Paul Brugger and his team for machining the parts. Finally, I would like to thank my family for their great support during this very busy semester.

Bibliography

[1] F. A. Delaloye, “Series elastic actuator (SEA): Laboratory set up for human augmentation and assistance,” Semester Project, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 2010.

[2] H. Vallery, J. Veneman, E. van Asseldonk, R. Ekkelenkamp, M. Buss, and H. van Der Kooij, “Compliant actuation of rehabilitation robots,” *IEEE Robotics Automation Magazine*, vol. 15, no. 3, pp. 60–69, Sep. 2008.

[3] G. A. Pratt and M. M. Williamson, “Series elastic actuators,” in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots'*, vol. 1, Aug. 1995, pp. 399–406.

[4] M. Zinn, B. Roth, O. Khatib, and J. K. Salisbury, “A new actuation approach for human friendly robot design,” *The International Journal of Robotics Research*, vol. 23, no. 4-5, pp. 379–398, 2004.

[5] *Maxon DC Motor RE 40 148877, Datasheet*, Maxon Motor, 2010.

[6] *Maxon Planetary Gearhead GP 52 C 223096, Datasheet*, Maxon Motor, 2010.

[7] *R+W Torque Limiter, Model ES2*, R+W Coupling Technology, 2010.

[8] S. Wolf and G. Hirzinger, “A new variable stiffness design: Matching requirements of the next robot generation,” in *Proc. IEEE International Conference on Robotics and Automation ICRA*, Pasadena, California, May 2008, pp. 1741–1746.

[9] *EPOS2 Positioning Controllers, Firmware Specification*, Maxon Motor, 2010.

[10] *EPOS2 Positioning Controllers, Getting Started*, Maxon Motor, 2010.

[11] *Maxon DC Motor and Maxon EC Motor, Key information*, Maxon Motor, 2010.

[12] *EPOS2 Positioning Controllers, Windows 32-Bit DLL*, Maxon Motor, 2009.

[13] *EPOS2 Positioning Controllers, Communication Guide*, Maxon Motor, 2010.

[14] G. Rouleau, *Simulink Real Time Execution*, The MathWorks, Jul. 2010. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/21908-simulink&reg-real-time-execution

[15] L. Daga, *Real-Time Blockset7.1 for Simulink*, The MathWorks, May 2007. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/3175-real-time-blockset-7-1-for-simulink

[16] S. Shapovalov, *RTsyncBlockset*, The MathWorks, Aug. 2009. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/24975-rtsync-blockset

[17] G. Vallabha, *Real-Time Pacer for Simulink*, The MathWorks, Oct. 2010. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/29107-real-time-pacer-for-simulink

[18] L. Righetti, J. Buchli, and A. J. Ijspeert, "Dynamic hebbian learning in adaptive frequency oscillators," *Phisica D*, vol. 216, pp. 269–281, 2006.

[19] J. Buchli, L. Righetti, and A. J. Ijspeert, "Frequency analysis with coupled nonlinear oscillators," *Phisica D*, vol. 237, pp. 1705–1718, 2008.

[20] L. Righetti, J. Buchli, and A. J. Ijspeert, "Adaptive frequency oscillators and applications," *The Open Cybernetics and Systemics Journal*, vol. 3, pp. 64–69, 2009.

[21] R. Ronsse, N. Vitiello, T. Lenzi, J. van den Kieboom, M. C. Carrozza, and A. J. Ijspeert, "Human-robot synchrony: Flexible assistance using adaptive oscillators," *IEEE Trans. Biomed. Eng.*, pp. 1558–2531, Oct. 2010.

[22] D. A. Winter, *Biomechanics and Motor Control of Human Movement*, 4th ed. New Jersey: Wiley, 2009.

[23] C.-C. Lin, M.-S. Ju, and C.-W. Lin, "The pendulum test for evaluationg spasticity of the elbow joint," *Arch Phys Med Rehabil*, vol. 84, no. 1, pp. 69–74, Jan. 2003.

[24] *Incremental Encoder RI 58-D / RI 58TD, Datasheet*, Hengstler, 2010.

A. Technical drawings

Files:

- `appendix\technical_drawings\ground_plate.jpg`
- `appendix\technical_drawings\ground_plate_drilling_list.jpg`
- `appendix\technical_drawings\bearing_pedestal.jpg`
- `appendix\technical_drawings\motor_fixture.jpg`
- `appendix\technical_drawings\shaft_motor.jpg`
- `appendix\technical_drawings\shaft_elbow.jpg`
- `appendix\technical_drawings\pendulum_fixture.jpg`
- `appendix\technical_drawings\pendulum.jpg`

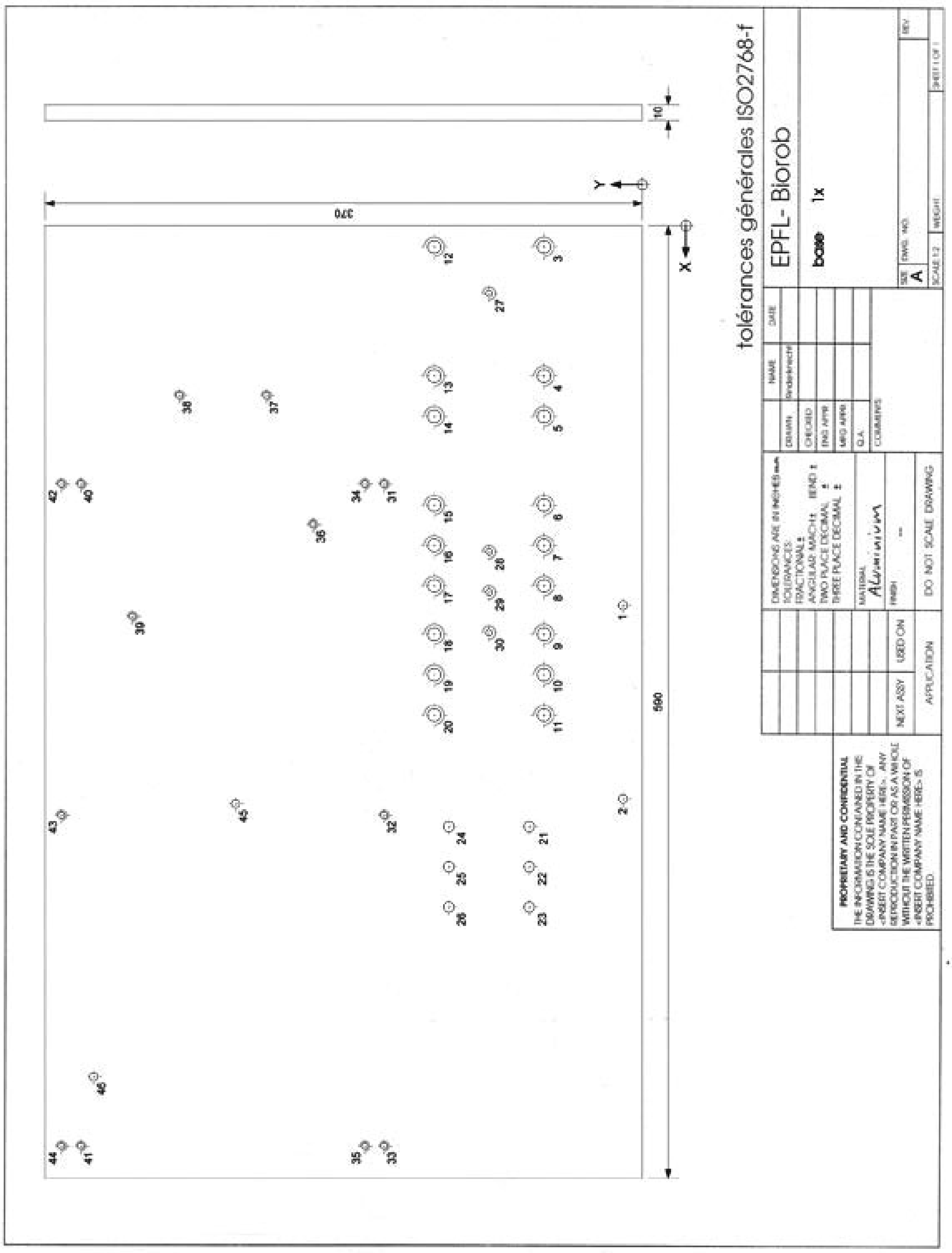
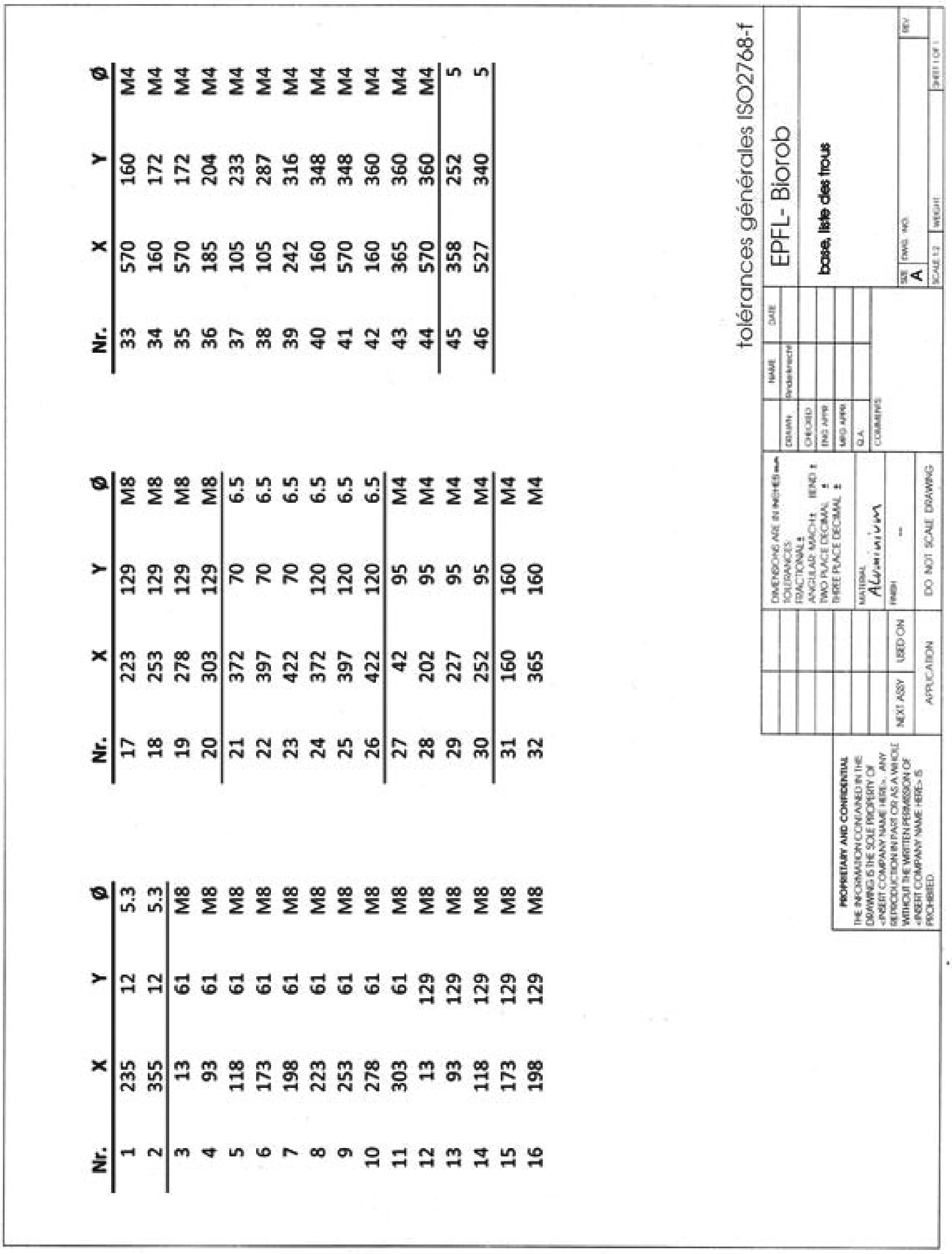


Figure A.1.: Ground plate. 1x.



Nr.	X	Y	Ø
1	235	12	5.3
2	355	12	5.3
3	13	61	M8
4	93	61	M8
5	118	61	M8
6	173	61	M8
7	198	61	M8
8	223	61	M8
9	253	61	M8
10	278	61	M8
11	303	61	M8
12	13	129	M8
13	93	129	M8
14	118	129	M8
15	173	129	M8
16	198	129	M8
17	223	129	M8
18	253	129	M8
19	278	129	M8
20	303	129	M8
21	372	70	6.5
22	397	70	6.5
23	422	70	6.5
24	372	120	6.5
25	397	120	6.5
26	422	120	6.5
27	42	95	M4
28	202	95	M4
29	227	95	M4
30	252	95	M4
31	160	160	M4
32	365	160	M4
33	570	160	M4
34	160	172	M4
35	570	172	M4
36	185	204	M4
37	105	233	M4
38	105	287	M4
39	242	316	M4
40	160	348	M4
41	570	348	M4
42	160	360	M4
43	365	360	M4
44	570	360	M4
45	358	252	5
46	527	340	5

tolérances générales ISO2768-f

NEXT ASSY | USED ON
APPLICATION

DIMENSIONS ARE IN INCHES mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
MATERIAL Aluminium
FINISH --
DO NOT SCALE DRAWING

	NAME	DATE
DRAWN	[illegible]	
CHECKED		
ENG APPR		
MFG APPR		
Q.A.		
COMMENTS		

EPFL- Biorob

base, liste des trous

SIZE A | DWG. NO. | REV
SCALE 1:2 | WEIGHT | SHEET 1 OF 1

Figure A.2.: Ground plate drilling list.

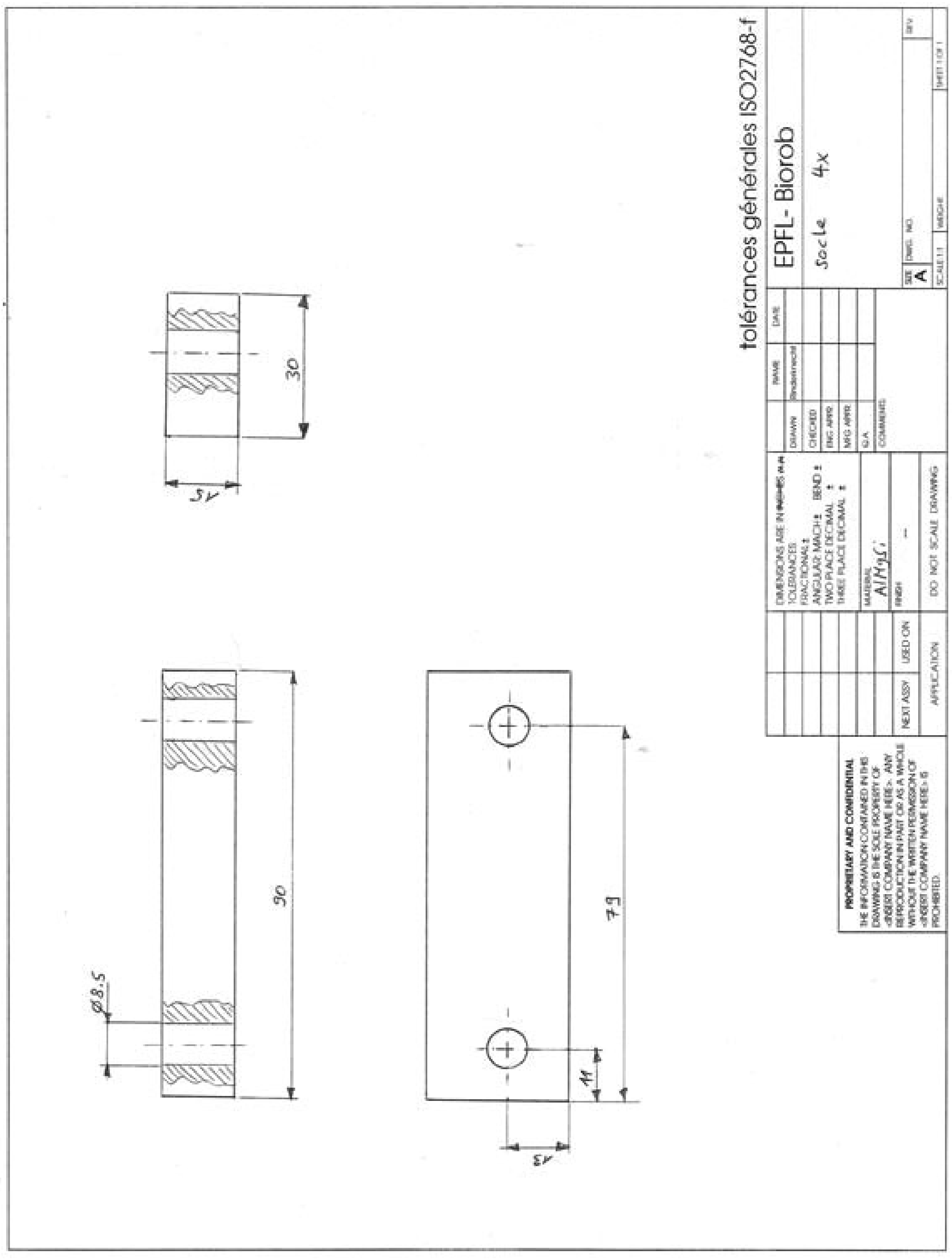


Figure A.3.: Pedestal base for bearings. 4x.

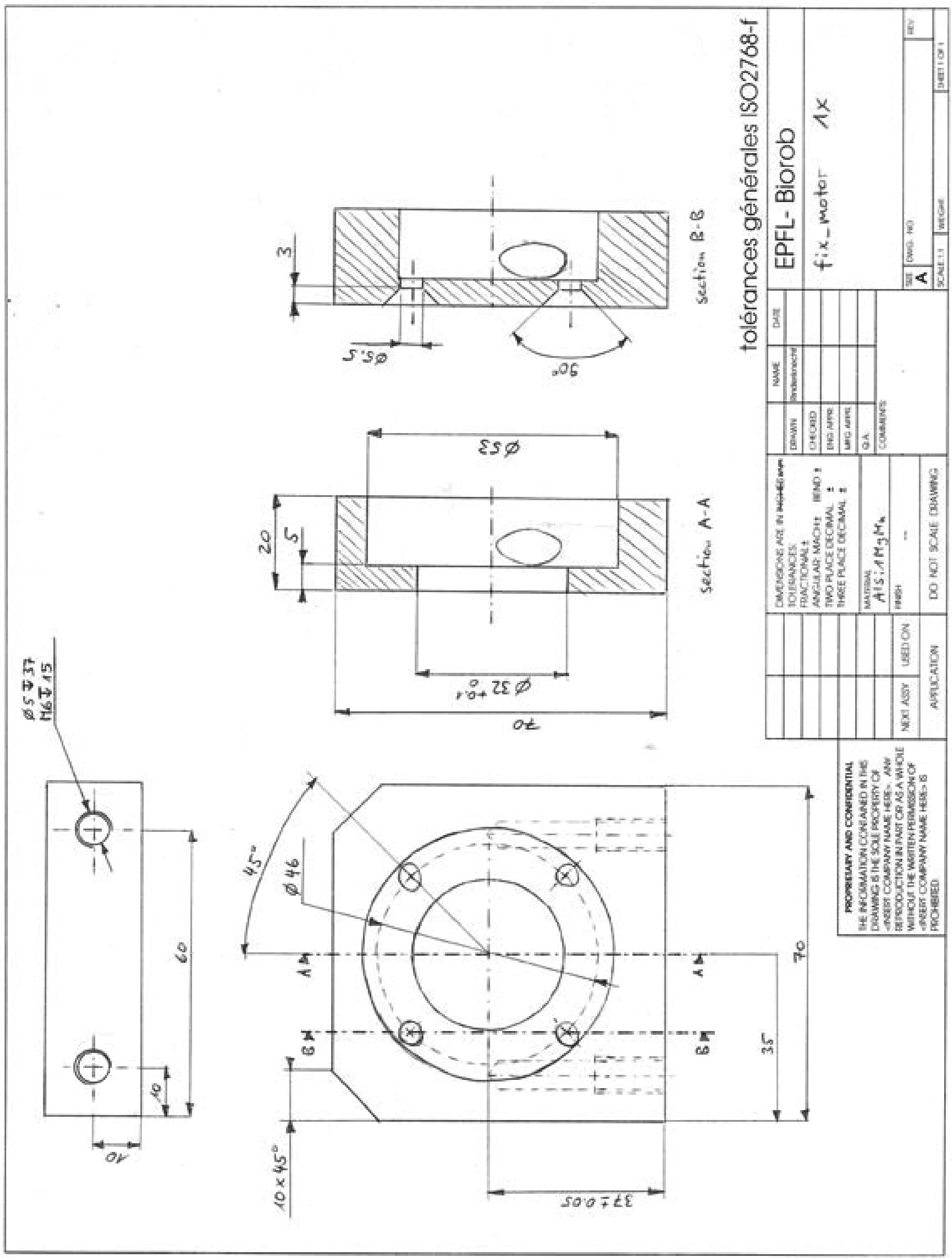


Figure A.4.: Motor fixture. 1x.

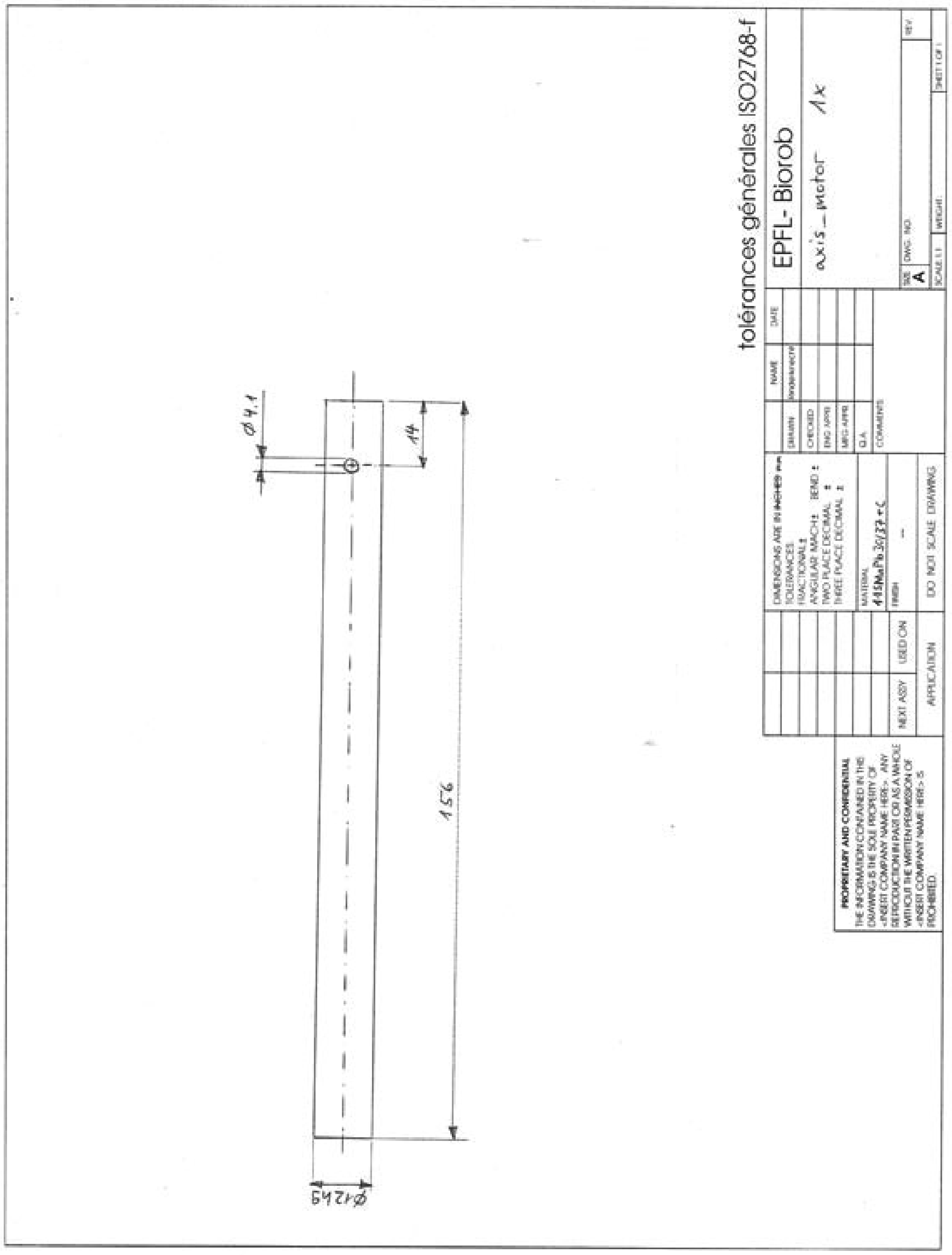


Figure A.5.: Motor shaft. 1x.

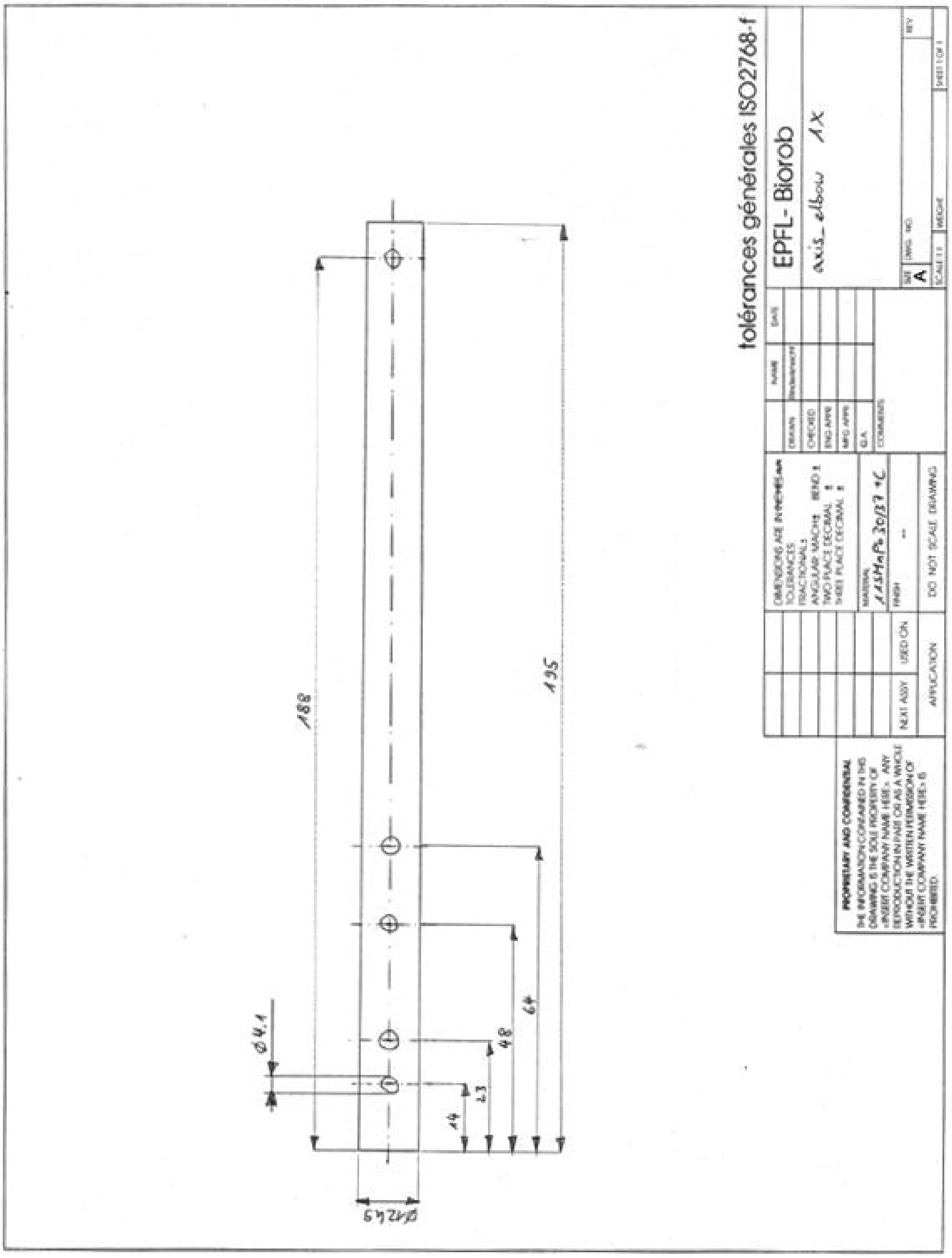


Figure A.6.: Elbow shaft. 1x.

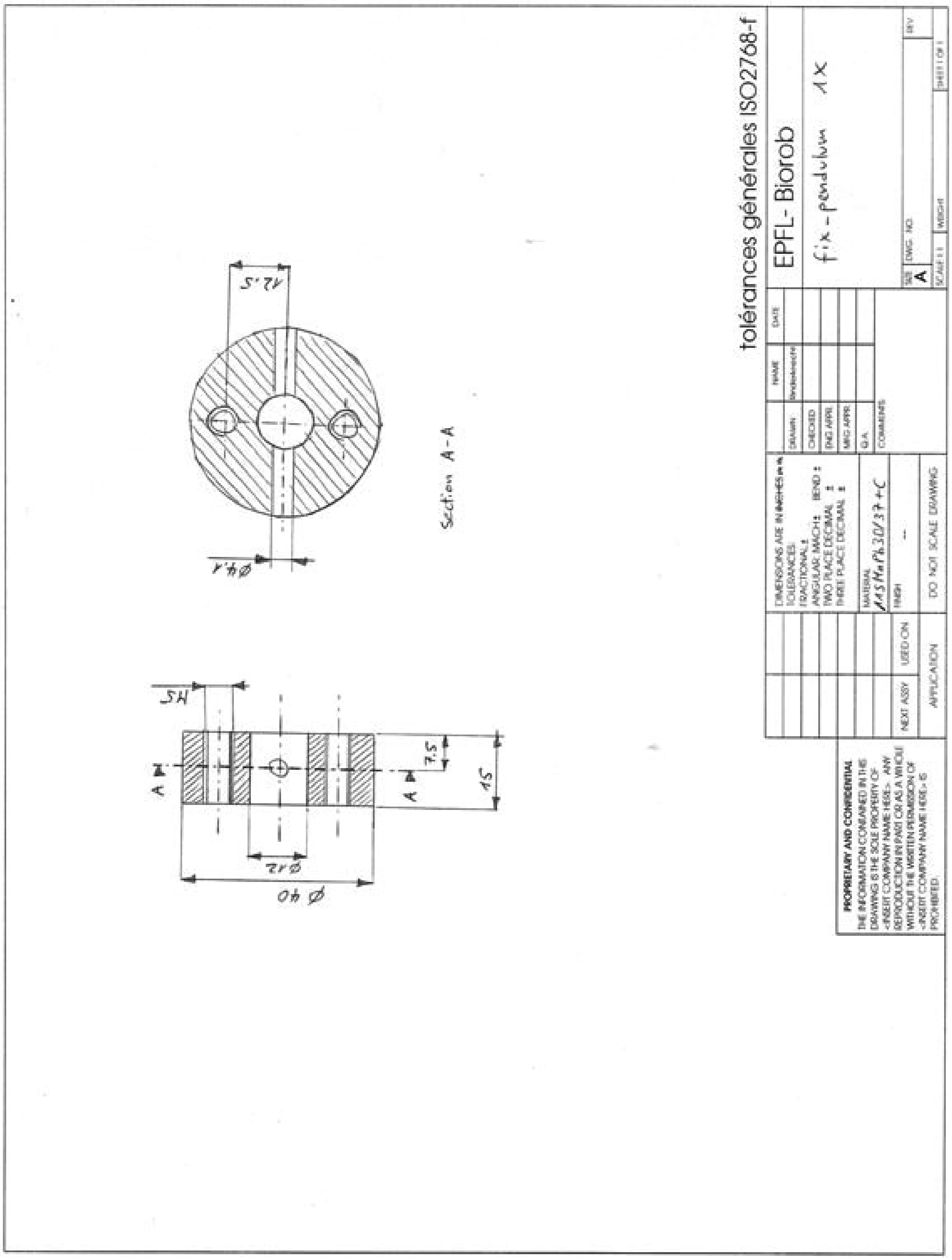


Figure A.7.: Pendulum fixture. 1x.

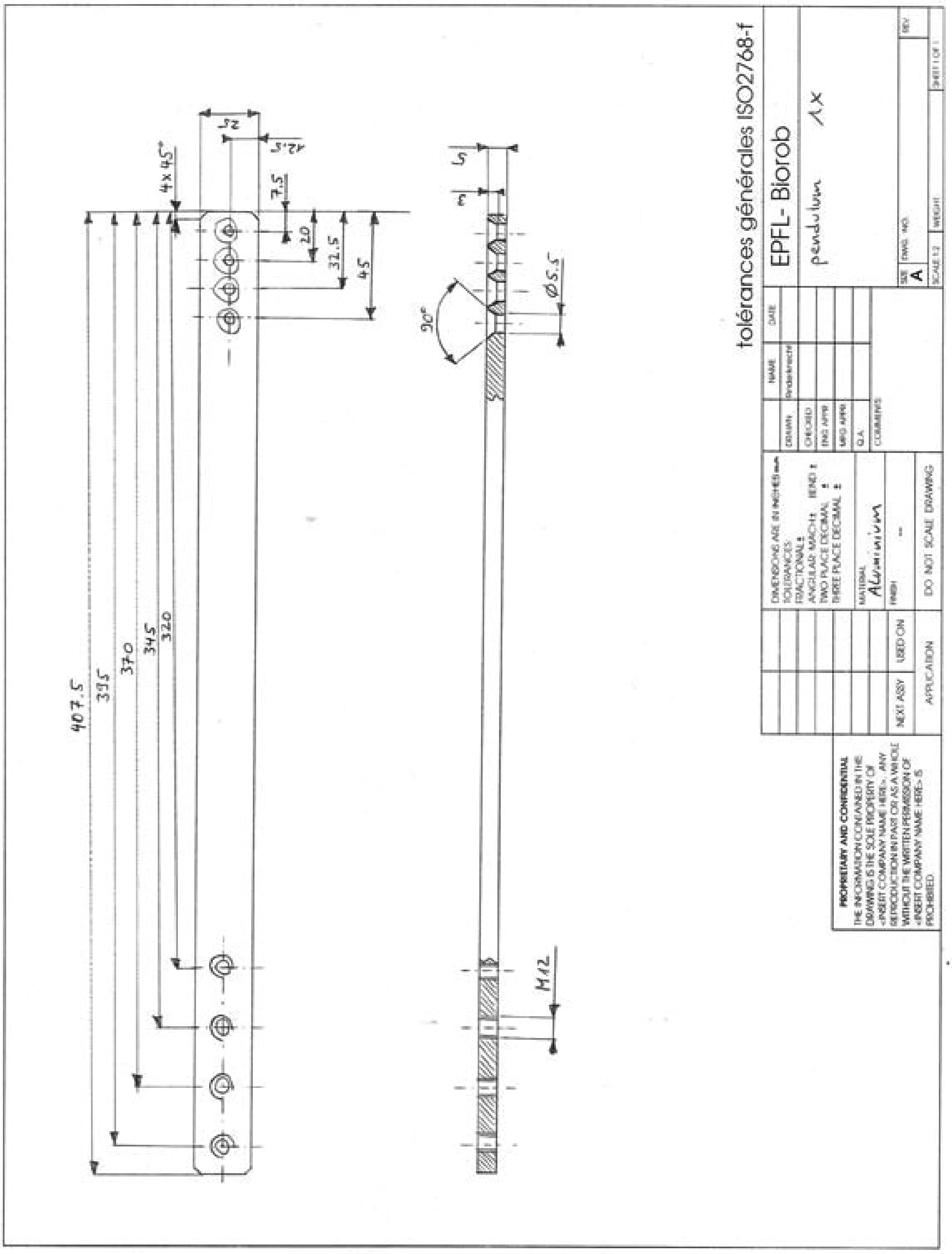


Figure A.8.: Pendulum. 1x.

B. Software

B.1. Spring dimensioning

Files:

- `appendix\spring_dimensioning\setup.exe`
- `appendix\spring_dimensioning\Durovis\Durovis.exe`
- `appendix\spring_dimensioning\Durovis\Durovis.LNG`
- `appendix\spring_dimensioning\Durovis\DUR_DRU.QSF`
- `appendix\spring_dimensioning\Durovis\DUR_SCH.QSF`
- `appendix\spring_dimensioning\Durovis\DUR_ZUG.QSF`
- `appendix\spring_dimensioning\Durovis\index.html`
- `appendix\spring_dimensioning\Durovis\df.htm`
- `appendix\spring_dimensioning\Durovis\sf.htm`
- `appendix\spring_dimensioning\Durovis\zf.htm`
- `appendix\spring_dimensioning\Durovis\DF_HELP.JPG`
- `appendix\spring_dimensioning\Durovis\DUR_DF.JPG`
- `appendix\spring_dimensioning\Durovis\DUR_SF.jpg`
- `appendix\spring_dimensioning\Durovis\DUR_ZF.jpg`
- `appendix\spring_dimensioning\Durovis\SF_HELP.JPG`
- `appendix\spring_dimensioning\Durovis\ZF_HELP.JPG`
- `appendix\spring_dimensioning\Durovis\unins000.exe`
- `appendix\spring_dimensioning\Durovis\unins000.dat`
- `appendix\spring_dimensioning\springs\1.dfb`
- `appendix\spring_dimensioning\springs\2.dfb`
- `appendix\spring_dimensioning\springs\3.dfb`

B.2. SEE dimensioning

Files:

- `appendix\see_dimensioning\see.m`
- `appendix\see_dimensioning\see_graph_K_R.m`
- `appendix\see_dimensioning\see_graph_L_F.m`
- `appendix\see_dimensioning\see_graph_L_K.m`
- `appendix\see_dimensioning\see_graph_L_R.m`

B.3. EPOS files

Files:

- `appendix\epos_studio\EPOS2 Project.pjm`

B.4. MATLAB complete example

Files:

- `appendix\example_matlab\EposCmd.dll`
- `appendix\example_matlab\Definitions.h`
- `appendix\example_matlab\stdint.h`
- `appendix\example_matlab\library_dll.m`
- `appendix\example_matlab\library_epos.m`
- `appendix\example_matlab\class_epos.m`
- `appendix\example_matlab\example_matlab.m`

B.5. Simulink complete example

Files:

- `appendix\example_simulink\EposCmd.dll`
- `appendix\example_simulink\Definitions.h`
- `appendix\example_simulink\stdint.h`

- `appendix\example_simulink\library_dll.m`
- `appendix\example_simulink\library_epos.m`
- `appendix\example_simulink\class_epos.m`
- `appendix\example_simulink\simulink_epos_library.mdl`
- `appendix\example_simulink\simulink_rehab_library.mdl`
- `appendix\example_simulink\simulink_epos_init.m`
- `appendix\example_simulink\simulink_epos_init_callback.m`
- `appendix\example_simulink\simulink_epos_exit.m`
- `appendix\example_simulink\simulink_epos_get_position.m`
- `appendix\example_simulink\simulink_epos_set_position.m`
- `appendix\example_simulink\simulink_real_time.m`
- `appendix\example_simulink\simulink_rehab_init.m`
- `appendix\example_simulink\example_simulink.mdl`

B.6. Signal logging example

Files:

- `appendix\example_logging\example_logging.m`

B.7. Experiments

Files:

- `appendix\experiments\EposCmd.dll`
- `appendix\experiments\Definitions.h`
- `appendix\experiments\stdint.h`
- `appendix\experiments\library_dll.m`
- `appendix\experiments\library_epos.m`
- `appendix\experiments\class_epos.m`
- `appendix\experiments\simulink_epos_library.mdl`

- appendix\experiments\simulink_rehab_library.mdl
- appendix\experiments\simulink_epos_init.m
- appendix\experiments\simulink_epos_init_callback.m
- appendix\experiments\simulink_epos_exit.m
- appendix\experiments\simulink_epos_get_position.m
- appendix\experiments\simulink_epos_set_position.m
- appendix\experiments\simulink_real_time.m
- appendix\experiments\simulink_rehab_init.m
- appendix\experiments\model.mdl
- appendix\experiments\create_training_protocol.m
- appendix\experiments\create_error_protocol.m
- appendix\experiments\save_data.m
- appendix\experiments\load_data.m
- appendix\experiments\clear_data.m
- appendix\experiments\plot_data.m
- appendix\experiments\process_data.m
- appendix\experiments\measured_sensitivity.m
- appendix\experiments\theoretical_torque.m
- appendix\experiments\theoretical_sensitivity.m
- appendix\experiments\data_participant_1.mat
- appendix\experiments\data_participant_2.mat
- appendix\experiments\data_participant_3.mat
- appendix\experiments\data_participant_4.mat
- appendix\experiments\data_participant_5.mat
- appendix\experiments\format_ticks.m

B.8. System identification

Files:

- `appendix\system_identification\fit.m`
- `appendix\system_identification\exp_cos.m`
- `appendix\system_identification\free_pendulum.mat`
- `appendix\system_identification\motor_pendulum.mat`
- `appendix\system_identification\device_data.txt`