

1 DESCRIPTION OF THE TASK

In order to allow for successful interplay of all elements developed within this project, project internally standardized interfaces for hardware and software have to be defined and enforced. The enabling subtasks of OBHP are:

- To provide discussion platforms and data sheet repositories for interface agreement, documentation and consultation
- To motivate and monitor adherence to the agreed interfaces from the earliest stage on of the project

Together with all project partners, OBHP will mediate the definition of:

- interfaces between data acquisition and signal processing packages
- interfaces between signal processing packages and prosthetic hardware

2 DESCRIPTION OF DELIVERABLE

This deliverable will define the following interfaces:

- interfaces between data acquisition and signal processing packages
- interfaces between signal processing packages and prosthetic hardware

These definitions will be crucial for successful interplay of all components.

3 IMPLEMENTATION & RESULT

The following interface templates provide the basis of what needs to be defined when a new interface is created. These guidelines will be adhered to within the project to unify interface descriptions.

3.1 TEMPLATE FOR INTERFACING DATA ACQUISITION WITH SIGNAL PROCESSING PACKAGES

3.1.1 PREREQUISITES

This deliverable provides a template for the data transmission of signals used in INPUT. The prerequisites for the application of this template is therefore the readily acquisitioned raw signals. This comprises:

- Analogue signal preprocessing (amplification, filtering, rectification,...)
- Data sampling by a control unit (microcontroller, DSP,...)

3.1.2 INTERFACE DESCRIPTION

An interface description regarding the transmission of data from an acquisition source to a signal processing unit has to detail the following interface descriptions:

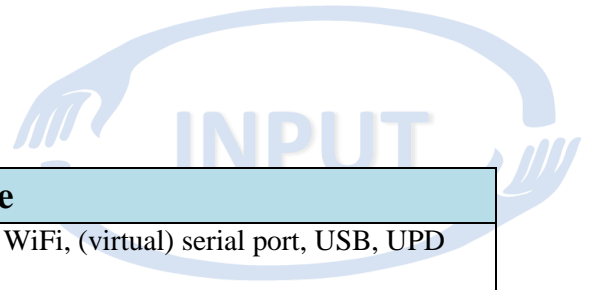


Table 1: Interface features to be described

Feature to be described		Example
Physical transmission medium		Bluetooth, WiFi, (virtual) serial port, USB, UPD port
Native sampling frequency		1 kHz, 1.8kHz, 9kHz,...
Transmission frequency		Baudrate, kB/s,
Transmission protocol		Event based, interrupt based, polling based
Data format		uint16, float, double
Conversion factors		to mVolts on skin, to degrees, to Newtons,...
Data endianness		little endian, big endian
Order of data transmission per frame		[length, ID1, ID2, ID3, payload, payload, payload, checksum]
Calculation of safety checks		Calculation of length, calculation of checksum (any/all safety checks are optional)
Number and order of information in payload		8 EMG, 3 IMU, 3 force sensors, ...
Available commands	Start of transmission	IDs, payloads, return values
	Stop of transmission	IDs, payloads, return values
	Configuration parameters	Sampling frequency, Amplification,...

3.1.3 PROPRIETARY INTERFACES

In case that the used physical data transmission underlies a proprietary protocol, the owner of this protocol has to provide the partners with an abstraction layer, which handles the communication in the background and provides the signals in a non-proprietary manner, which then has to be described with the same definitions as described in Section 3.1.2.

3.1.4 EXEMPLARY INTERFACE DESCRIPTION OF OBHP EMG SIGNAL INTERFACE: UDP SOCKET

The data transfer protocol of Otto Bock prostheses is proprietary and therefore an abstraction layer is offered to send and receive data to and from the prostheses controller. The abstraction layer is implemented as .NET DLL package, which can be directly integrated .NET frameworks. It further comes with a standalone application GUI, which handles data communication and exposes the communication interface via a UDP socket. It can therefore be accessed by any type of program capable of reading and writing from/to UDP sockets (Matlab, C/C++, Python, etc.).

Via this connection, data can be streamed from an Ottobock Michelangelo prosthesis to a host program (signal processing unit).

Installation

It should be checked that the system number format is set to use the “.”- as a decimal point; and not “,” (otherwise the calibration files will fail to load). This can be checked easily by going to the “Control

Panel” and then to “Change date, time or number formats” and then under *Additional Settings* as shown in **Figure 1**.

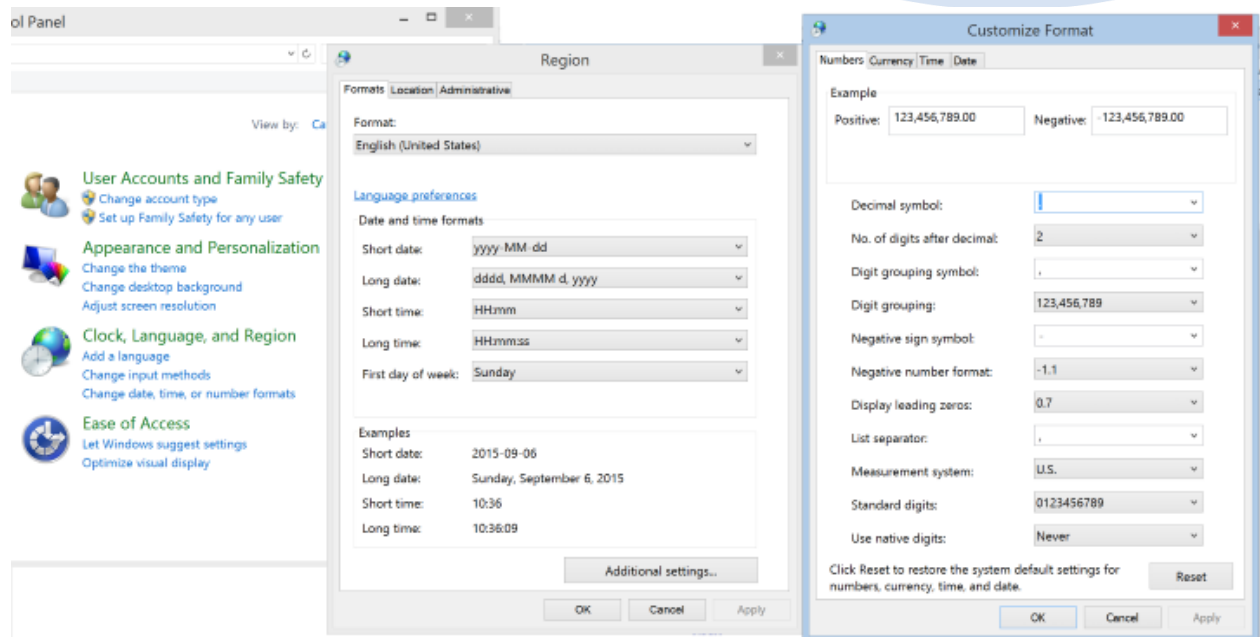
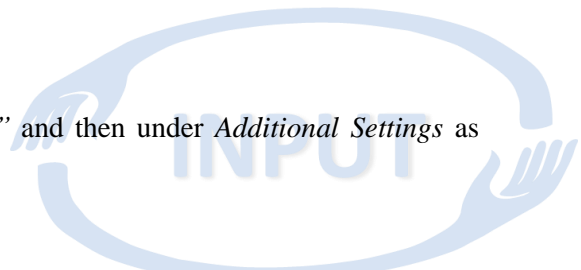
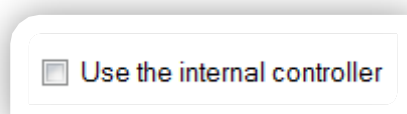


Figure 1: Checking the System Number Format

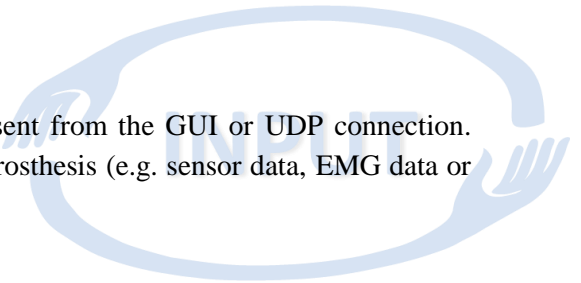
Operation modes and prosthesis types

Prosthesis description: Gripper + Wrist Unit: This prosthesis permits grasping using Palmar (or Fine Pinch) and Lateral (or Key Grip) grasp. It has a wrist unit with 2 full degrees of freedom with which the hand can perform Pronation, Supination, Flexion and Extension.

1. There are two different types of USB Bluetooth dongles available, namely, *FAST* (*BaudRate: 460800*) and *SLOW* (*BaudRate: 115200*). In INPUT, only the 460800 version is used. Further, the prosthetic hands are fitted with *controllers* that support transmission at either 100 Hz or 1000 Hz. In INPUT, only the 1000Hz version is used. The SLOW-Dongle can only connect with the 100Hz-Controller. The FAST-Dongle can connect with both the 100Hz and 1000Hz-Controller.
2. Internal Controller: The controller connected with the prosthesis supports the *standard* state-of-the-art (SoA) control as programmed by the therapist via the OttoBock AxonSoft software package. This mode may be used when the SoA control algorithm is needed for comparative measurements. This mode is activated by checking the “Use the internal controller” checkbox on the GUI, before pressing the *Connect* button.



In this mode, the prosthesis will not respond to commands sent from the GUI or UDP connection. However, at 100 Hz the sensor data can be logged from the prosthesis (e.g. sensor data, EMG data or controller-state).



Connecting to the prosthesis

1. Turn ON the prosthesis with Bluetooth enabled:

Press the button on the prosthesis for about 5 seconds. First "...beep, beep...", then after a pause two more signals (like: "...beep, beep,..."). The LED will be continuously blue. A very short press indicates the battery state, a slightly longer press will turn the controller off.

2. Open the GUI:

Start the application GUI "MichelangeloGUI.exe (**Figure 2**). To connect with the prosthesis, follow the steps below:

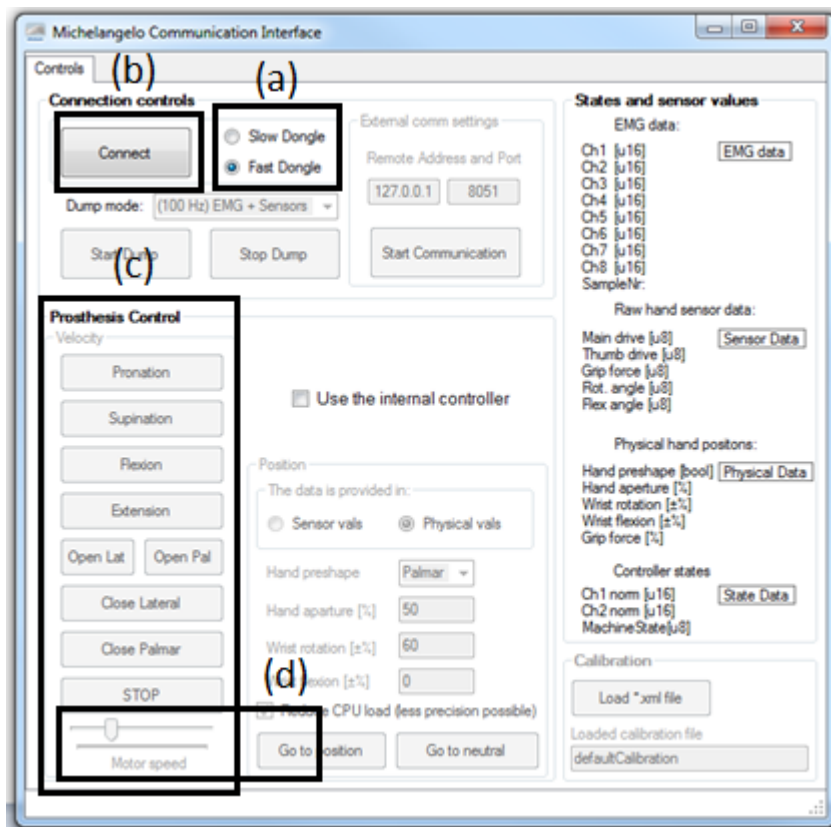


Figure 2: The view of the GUI when started

- a. Connect the Bluetooth Dongle to your computer. Check the 'Fast Dongle' RadioButton.
- b. Then press the "Connect" button and wait until the program connects to the Prosthesis. The text of the "Connect" button will change to "Disconnect" and hereafter the same button shall be used to disconnect the prosthesis from the program when finished.

- c. The panel on the left hand side in Figure 2, marked as (c), can be used to send ‘velocity commands’ to the prosthesis. You can control the velocity (or speed) of any motor using the slider “Motor speed” (represented via the box (d)).

Now the prosthesis can be moved by clicking buttons. For example, if the *Motor Speed* slider is set to 20% and the *Pronation* button is pressed, then the prosthesis will start pronating with 20% of the maximum motor speed. To stop pronating, click on the ‘STOP’ button.

3. Loading the Calibration File:

The prosthesis sends sensor data to the GUI, but this sensor data is ‘raw’. The calibration file is used to convert raw sensor data to human readable normalized sensor data (Table 1, Conversion factors). To load the calibration file click on the “Load *.xml file” (see **Figure 3**). Under the file path .../ MichaelAngeloHand/Calibration/..., you will find four Calibration files. Select the appropriate file name based on the prosthesis you have and click Open. In case that you don’t care for the position control or sensor data (e.g., you just want to read the EMG, or to use velocity control, use the defaultCalibration.xml)

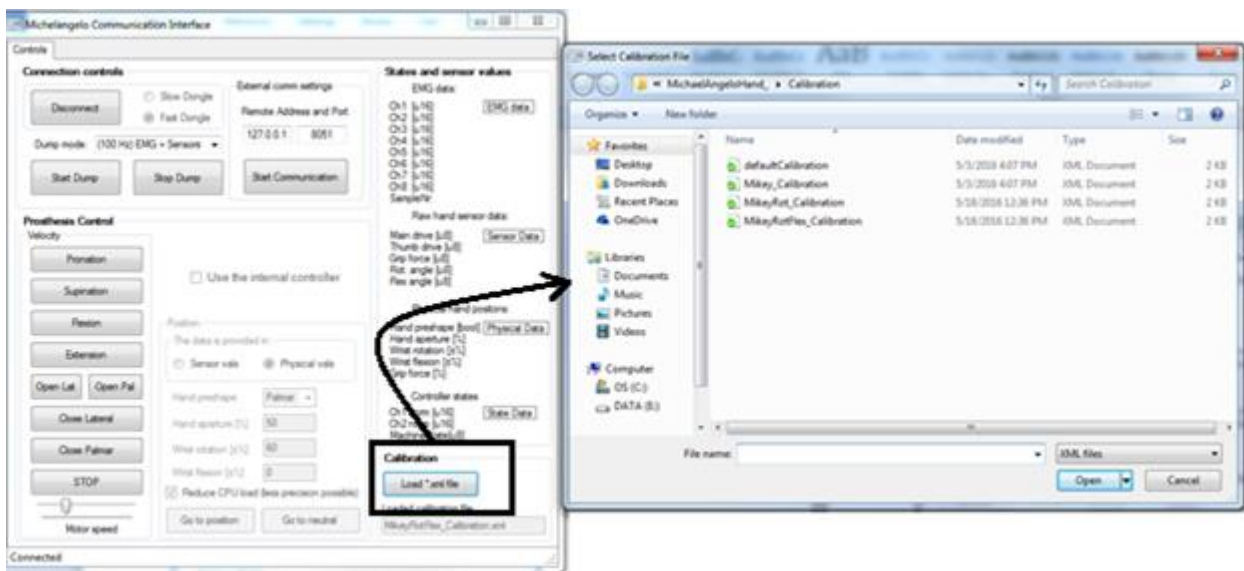


Figure 3: Loading the Calibration File

4. Configure the DUMP Mode:

Under the *Connect/Disconnect* button, there is a *drop-down* menu labeled “Dump Mode”. There are two dump modes available as shown in Figure 4. Choose the (1KHz) EMG mode. This dump mode will transmit EMG data only at a rate of 1kHz. This mode is useful when one needs to process raw EMG signals. In this dump mode only velocity control of the prosthesis is possible (which represents the case of conventional prosthetic control).

- a. To start the dump, select the required dump mode and click “Start Dump”.

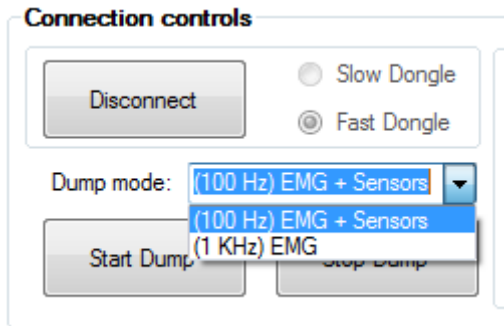
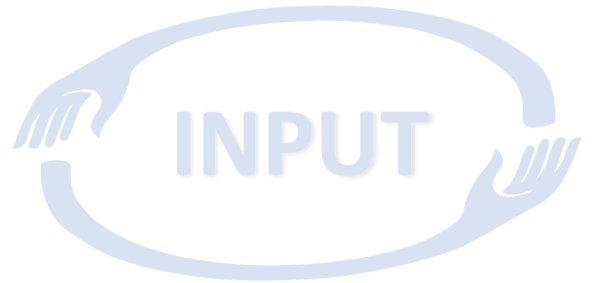


Figure 4: Dump Modes

Interfacing with the prosthesis from an external program

The biggest advantage of this interface is that it allows controlling and reading the data from the prosthesis via a UDP port opened in an external program (e.g. Matlab, C/C++, Python, etc.). To establish a UDP connection from an external program, proceed as follows:

- a. In the “External comm settings” group-box enter the (IP address, Port) at which you would like to send commands from your external code. As shown in **Figure 5**, the default IP and listening UDP port are set to 127.0.0.1 and 8051; it is advisable/convenient to directly use these settings.
- b. Click on the “Start Communication” button, to control the hand from your external program (written in Matlab, C/C++, Python, etc.). Movement commands can be sent to the hand as well as EMG and sensor data can be read in.

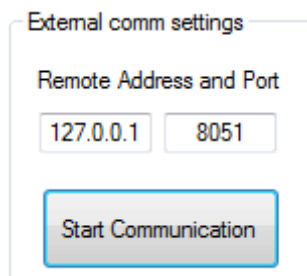


Figure 5: External Communication Settings.

If EMG data should be transmitted from the prosthesis (or sensor data for experimental reasons, e.g., grasping force, hand position), click on the “Start Dump” button. The interface will send the data on port 127.0.0.1/8052 (i.e. your program should listen for sensor data UDP packets at port 8052). Generalized, the sending port of the GUI (the one the host application should listen to) is always +1 of the listening port of the GUI (the sending port of your application)

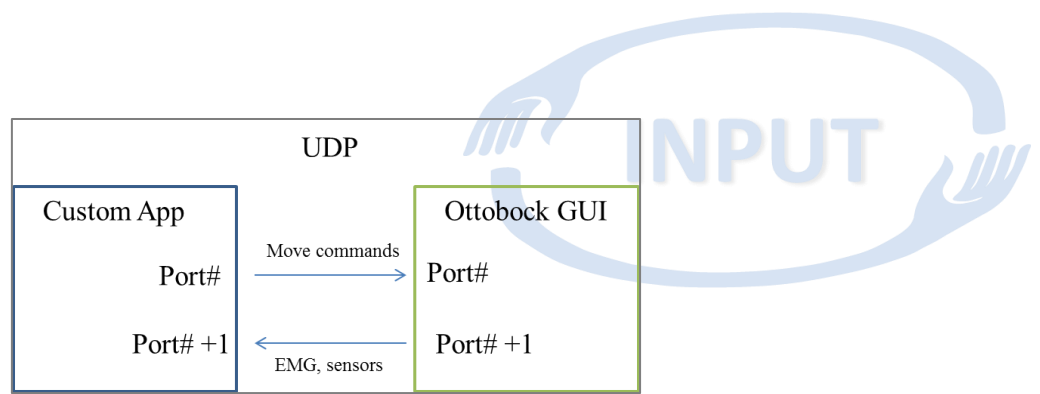


Figure 6: Port allocation for sending and receiving data from the Ottobock GUI to a custom application, such as a signal processing unit

The Packet Coding Scheme:

1. For the Receiver

a. For the Dump mode (*1kHz*) *EMG data*: we get 18 bytes in a single UDP packet. **This mode should be used in INPUT.**

i. byte0 and byte1 = 1st channel of EMG to convert in Uint16, range [0, 856]

ii. byte2 and byte3 = 2nd channel of EMG to convert in Uint16, range [0, 856]

...

iii. byte14 and byte15 = 8th channel of EMG to convert in Uint16, range [0, 856]

iv. byte16 = Counter

For sake of completeness also the other receiver mode is described here: (available but not to be used in INPUT)

b. For the dump mode (*100Hz*) *Sensor + EMG Data*: we get 35bytes is a UDP packet.

____raw sensor data____

i. byte0 = Main Drive in Uint8, range [0, 255]

ii. byte1 = Thumb Drive in Uint8, range [0, 255]

iii. byte2 = Rotation Angle in Uint8, range [0, 255]

iv. byte3 = Flexion Angle in Uint8, range [0, 255]

v. byte4 = Force in Uint8, range [0, 255]

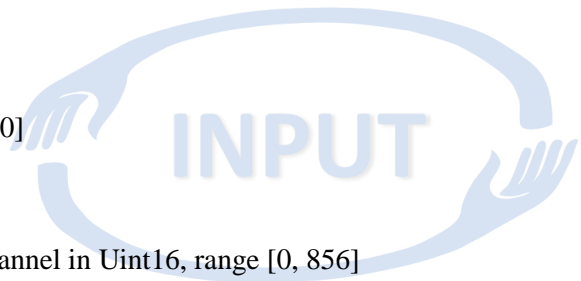
____normalized sensor data (human readable) ____

vi. byte5 = Grasp Type in Int8, 0 for palmar and 1 for lateral

vii. byte6 = Aperture in Int8, range [-100, 100]

viii. byte7 = Pronation/Supination in Int8, range [-100, 100]

ix. byte8 = Flexion/Extension in Int8, range [-100, 100]



x. byte9 = Force in Int8, range [-100, 100]

____Controller State, when in Co-Contraction Mode____

xi. byte10 and byte11 = First Control Channel in Uint16, range [0, 856]

xii. byte12 and byte13 = Second Control Channel in Uint16, range [0,856]; These last two are normalized EMG

xiii. byte14 and byte15 = Machine State in Uint16

____Raw EMG data____

xiv. byte16 and byte17 = 1st channel of EMG to convert in Uint16, range [0, 856]

xv. byte18 and byte19 = 2nd channel of EMG to convert in Uint16, range [0, 856]

...

xxi. byte30 and byte31 = 8th channel of EMG to convert in Uint16, range [0, 856]

____Overhead for data management____

xxii. byte32 and byte33 = two counters

xxiii. byte34 = Boolean which says if the hand has reached the command position

Table 2: Interface description table for Ottobock GUI

Feature		Value	
Physical transmission medium		Bluetooth (proprietary), then UDP port	
Native sampling frequency		1 kHz	
Transmission frequency		UDP packaging	
Transmission protocol		Polling based	
Data format		double	
Conversion factors		Provided in config files	
Data endianness		Big endian	
Order of data transmission per frame		See above	
Calculation of safety checks		none	
Number and order of information in payload		8 EMG (chan1 to chan8)	
		Parameters	Return values
Available commands	GUI buttons	See above	See above



3.2 TEMPLATE FOR INTERFACING SIGNAL PROCESSING PACKAGES WITH PROSTHETIC HARDWARE

3.2.1 PREREQUISITES

This section describes how a custom application can interface with a physical prosthesis to control it. The prerequisites for this template to be applicable are therefore:

- An application (signal processing unit) exists which has already computed the instantaneous movement commands for the prosthesis
- A prosthesis is connected to the host of the processing unit

3.2.2 INTERFACE DESCRIPTION

An interface description regarding the control commands of an application to control a has to detail the following interface descriptions:

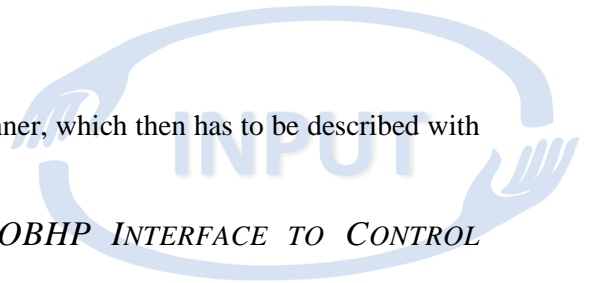
Table 3: Interface features to be described

Feature to be described		Example
Physical transmission medium		Bluetooth, WiFi, (virtual) serial port, USB, UPD port
Frequency of control signals sent		Fixed rate, only on change of values
Data format		uint16, float, double
Conversion factors		Angular speed, motor torque, degrees, Newtons,...
Data endianness		little endian, big endian
Order of data transmission per frame		[length, ID1, ID2, ID3, payload, payload, payload, checksum]
Calculation of safety checks		Calculation of length, calculation of checksum (any/all safety checks are optional)
Number and order of information in payload		Motor 1, Motor 2,...
Available commands	Start of transmission	IDs, payloads, return values
	Stop of transmission	IDs, payloads, return values
	Configuration parameters	Speeds, stop values,...

3.2.3 PROPRIETARY INTERFACES

In case that the used physical data transmission underlies a proprietary protocol, the owner of this protocol has to provide the partners with an abstraction layer, which handles the communication in the

background and provides the signals in a non-proprietary manner, which then has to be described with the same definitions as described in Section 3.2.2.



3.2.4 EXEMPLARY INTERFACE DESCRIPTION OF OBHP INTERFACE TO CONTROL PROSTHESIS

The interface described above in Section 3.1.4 also allows to control the OttoBock Michelangelo hand via the standard UDP connection. UDP is a very simple networking protocol that allows transmission of data packets (in this context, commands to/from the hand) between two applications, either, on same or different PCs. The Michelangelo hand can perform different movements like *Grasping* (either with Lateral or Palmar grasp), *Wrist Rotation* (both Pronation and Supination) and *Wrist Flexion/Extension*. To accomplish these movements, the prosthesis has different on-board motors, which move the motor-shaft/actuator to the commanded position.

With this interface, the Michelangelo hand can be controlled in two different modes,

- *Velocity control*: This mode allows controlling the velocity of individual motors i.e. closing/opening velocity of grasping or rotation velocity for Pronation/Supination.
- *Position Control*: This mode allows controlling the position of individual motor-shafts i.e. it allows you to pre-shape a grasp with a specific amount of closure or rotate to a specific angle w.r.t the neutral position.

The Michelangelo hand also has on-board sensors. With this interface, it is also possible to receive the sensor data from prosthesis. The interface provides data from the following prosthesis sensors,

- *EMG*: Up to 8 EMG sensors can be connected to the prosthesis. The interface can stream EMG data at either 100 Hz or 1000 Hz.
- *Force*: The grasping force exerted on any grasped object can be measure by using the on-board force sensor.
- *Actuator Position*: The rotation and flexion angles w.r.t the neutral position are available via the interface.

Additionally one more control mode is available, which does not rely on any external software (all computations are done in prosthesis as in a conventional control case)

- *Internal controller states*: The states of the embedded prosthesis controller that is used to drive the prosthesis as programmed by the therapists.¹

1. Controlling Position of the Hand (available only if the 100 Hz mode is selected, the Dump is started and if the option “use internal controller” is not checked):

The *position group-box* in the GUI is used to control the position of individual motors on the prosthesis (position group-box of the GUI has been shown in **Figure 7**). This option allows you to control either the ‘Sensor vals’ or the ‘Physical vals’. The Physical values are easy to interpret for humans, as they represent normalized sensor value in range [-100, 100]. In the position group-box,

¹ Due to software incompatibility, only the default parameters of the prosthesis controller can be used. AxonSoft cannot be used to modify the control parameters, as would be possible with a regular prosthesis.

you can set the grip-type (either Palmar or Lateral), the hand aperture (in range [0, 100]) and the wrist position as:

- Wrist rotation [$\pm\%$]: where, a positive position value in range [0, 100] represents Pronation (clockwise displacement from 0) and a negative value in range [-100, 0] represents Supination (anti-clockwise displacement from 0).
- Wrist flexion [$\pm\%$]: where, a positive position value in range [0, 100] represents Extension and a negative value in range [-100, 0] represents Flexion (where, -100 means full flexion of the wrist).

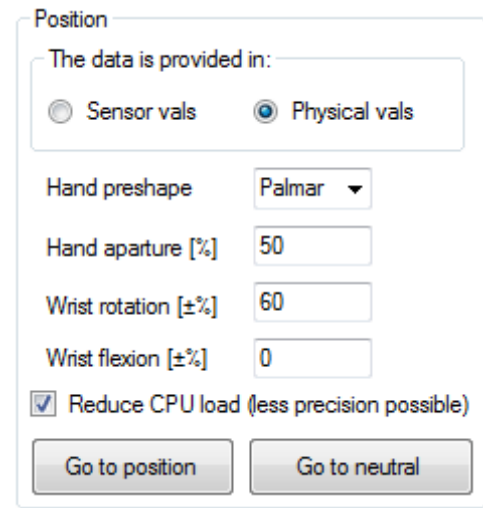


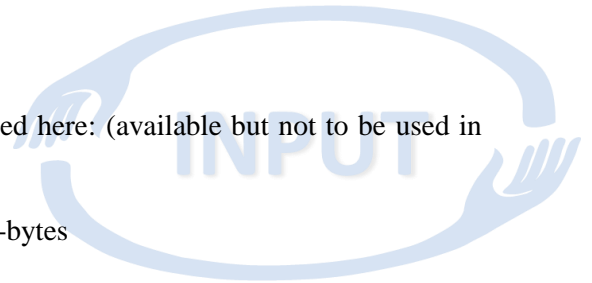
Figure 7: Position group-box in the GUI used to control the position of individual prosthesis motors.

2. For the Sender (i.e. to control the prosthesis via UDP packets):

Three modes are supported as follows,

- a. "*Velocity Mode*": it has 9 unsigned-byte: **This mode should be used in INPUT!**
 - i. byte0 = 1: to indicate velocity-control mode
 - ii. byte1 = Palmar Grip Closing command in range [0, 255]
 - iii. byte2 = Palmar Grip Opening command in range [0, 255]
 - iv. byte3 = Lateral Grip Closing command in range [0, 255]
 - v. byte4 = Lateral Grip Opening command in range [0, 255]
 - vi. byte5 = Pronation Velocity in range [0, 255]
 - vii. byte6 = Supination Velocity in range [0, 255]
 - viii. byte7 = Flexion Velocity in range [0, 255]
 - ix. byte8 = Extension Velocity in range [0, 255]

For sake of completeness also the other 2 modes are described here: (available but not to be used in INPUT)



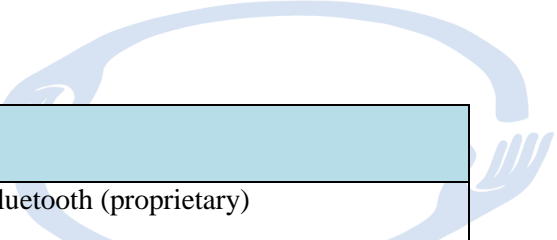
- b. "*Position Control Mode*": it has 5 or 8 signed-bytes
 - i. signed_byte0 = 2: to indicate position-control mode
 - ii. signed_byte1 = Grip Type, 0 - Palmar, 1 - Lateral
 - iii. signed_byte2 = Grip Closure in range [0, 100]
 - iv. signed_byte3 = Supination/Pronation, range [-100, 100]
 - v. signed_byte4 = Flexion/Extension, range [-100, 100]

If there are only 5 bytes, then the speed will be always maximal. Otherwise:

- vi. signed_byte5 = Maximum speed of the Grip, range [0, 100]
 - vii. signed_byte6 = Maximum speed of the Rotation, range [0, 100]
 - viii. signed_byte7 = Maximum speed of the Flexion, range [0, 100]
- c. "Neutral Position": The hand will come back to the neutral position. It has only one byte
 - i. Byte0 = 0

3.3 SAMPLE IMPLEMENTATION OF THE OTTOBOCK INTERFACE

A sample python script "MHand_Python.py" is available, implementing the host app functionalities using the Ottobock GUI. This script allows sending triangular/square waves to the prosthesis in both velocity and position mode. It also receives data back from the prosthesis and plots the results. If you do not have experience with python, first, install *Anaconda with Python 3.5* (www.continuum.io/downloads). After the installation is complete, open *Spyder* it is a very simply IDE for Python programming (just like the editor in Matlab or Visual Studio for C/C++). Open the python script in *Spyder* and run the program (after appropriately connecting the prosthesis with the provided interface).



Feature		Value
Physical transmission medium		UDP, then Bluetooth (proprietary)
Frequency of control signals sent		only on change of values (prosthesis moves with last command received as long as no other value is received)
Data format		uint16
Conversion factors		motor torque
Data endianness		big endian
Order of data transmission per frame		See details above
Calculation of safety checks		none
Number and order of information in payload		See details above
Available commands	Ottobock GUI	See details above

4 SUBCONTRACTING

No subcontracting was needed – all work was done by OBHP.