IH2 Azzurra Series

Self-Contained Robotic Hand *Basic* User Guide

Version 1.7

www.prensilia.com

Summary

Summary 3
Welcome to Prensilia! 5
1 Usage and safety considerations 5
1.1 Usage conditions 5
1.2 Research applications (examples) 5
1.3 Technical support and repairs 5
2 Introduction 6
2.1 Mechanisms overview 7
2.2 Sensory system overview 8
2.2.1 Motor rotary encoders 8
2.2.2 Motor current sensors 8
2.2.3 Tendon tension sensors[2] 8
2.2.4 Proximity sensors 9
2.2.5 External analog sensors 9
2.2.6 Sensor data resolution and encoding table 9
2.3 Control system overview 9
2.3.1 Calibration routines 10
2.3.2 Automatic grasps 11
2.4 Communication protocol overview 14
2.4.1 RS232 (COM port) communication settings 14
2.4.2 Addresses of the motors 14
2.4.3 STATUS byte and control modes 14
2.4.4 Data binary encoding 15
2.4.5 Read delays 16
3 Installation 17
3.1 Install the USB cable 17
3.2 Connect the cables and power-ON the hand 18
4 HLHC commands 19
4.1 Single DoA commands 19
4.1.1 MoveMotor 19
4.1.2 SetFingerPosition 19
4.1.3 SetFingerForce 20
4.1.4 SetFingerCurrent 20
4.1.5 SetFingerCurrPos 21
4.1.6 GetFingerPosition 22
4.1.7 GetFingerForce 22
4.1.8 GetMotorCurrent 23
4.1.9 GetExternalSensor 23
4.1.10 LowLevelVersion 23
4.2 Status/Calibrate commands 23
4.2.1 GetFingerStatus 23
4.2.2 FirstCalibration 24
4.2.3 FastCalibration 25
4.3 Whole hand commands 26
4.3.1 HighLevelVersion 26
4.3.2 StopALL 26
4.3.3 SetHandPosture 26
4.3.4 OpenALL 27
4.3.5 EnableStreaming 27
4.3.6 DisableStreaming 28
4.3.7 Automatic Grasps 28

4.3.8 Grasp Stepper 31
4.4 HLHC memory commands 32
4.4.1 MemCylPreShape 32
4.4.2 MemTriPreShape 33
4.4.3 MemBiPreShape 33
4.4.4 MemLatPreShape 33
4.4.5 MemLowCurr - I_L 34
4.4.6 MemHighCurr - I_H 35
5 LLMC commands 36
5.1.1 LLMC command description 37
5.1.2 LLMC commands chart 38
5.1.3 Command data bytes format table 38
6 Demo application 39
7 Resuming tables 42

Welcome to Prensilia!

Get ready to experience the advanced motor and sensory features of your Prensilia IH2 Azzurra Series Hand. Setting up your hand is easy: simply connect the USB plug provided to your host controller and power on the hand with an external supply. Take some time to explore the features on your device. This guide provides instructions and tips to help you learn the basic quickly, to properly operate your hand and to replace its parts. Thanks for acquiring this prototype! We hope it will boost your research!

⚠ This is the *basic user guide* freely downloadable from Prensilia.com. It does not contain chapters 7 (Maintenance) and 8 (Advanced users) and pictures showing technical details. All these information are available on request.

1 Usage and safety considerations

1.1 Usage conditions

- The robotic hand you purchased is a **prototype** for research purposes only.
- The prototype should be used and experimented for research only in **laboratory**.
- Do not use the prototype in any system on which people's lives depend (life support, weapons, etc.).
- In the case of human or animal experimentation make sure to have the supply and the connections from/to the prototype electrically separated from the animal/human.
- Always connect the prototype to a host PC, through the USB cable provided.
- The prototype is not water-proof: keep it away from water or other liquids.
- Do not insert any object in the hollows of the palm cover.
- If an Institutional Review Board approval (or other kinds of permission) is required by your Institution to carry out experiments with a non FDA, non CE approved prototype, you should obtain it before starting your experiments.
- Use a proper power supply: do not supply the prototype with electrical levels not compliant with the ratings described in this guide.
- The electronic boards of the prototype contain components that are sensitive to electro- static-discharge. Care should be taken when handling the hand without the cover.
- Prensilia S.r.l. does not warrant for the research usage that you may carry out, and for any other usage outside from the laboratory, or not complying with this user guide.

1.2 Research applications (examples)

- Prosthetics.
- Humanoid, industrial, cognitive, rehabilitation robotics.
- Assistive robotics and technology.
- Tele-control (remote grasping and manipulation).
- Human-robot interaction.
- Brain-Machine Interfaces (BMI).
- Bidirectional Human-Machine Interfaces (HMI).
- Neuroscience (sensorimotor control of the hand).
- Education and performing arts.

1.3 Technical support and repairs

Prensilia S.r.l. provides technical and user support via email (support@prensilia.com). Support is provided for the lifetime of the equipment. Requests for repairs should be made through the Support department. For damage occurring outside of the warranty period or provisions (see the Agreement), customers will be provided with cost estimates prior to repairs being performed.

2 Introduction

IH2-Azzurra hand prototype is a five-fingered, self-contained anthropomorphic hand with five intrinsic electrical motors. Each motor activates one Degree of Actuation (DoA); in particular:

- one for the flexion/extension of the thumb;
- one for the flexion/extension of the index;
- one for the flexion/extension of the middle;
- one for the joined flexion/extension of the ring-little fingers; and
- one for the abduction/adduction of the thumb.

Each finger has two phalanxes connected by means of two joints: the proximal phalanx (the closest to the palm) is made of aluminum whereas the distal phalanx (that includes the fingertip) is made of rubber. Each finger is actuated by means of a tendon that runs inside the phalanxes and is wrapped around the pulleys in the joints. While closing around an object the finger automatically adapts on it. This architecture promotes stability during precision or power grasps, which are by definition those grasps achieved by means of fingertips or palmar areas of the hand, respectively.

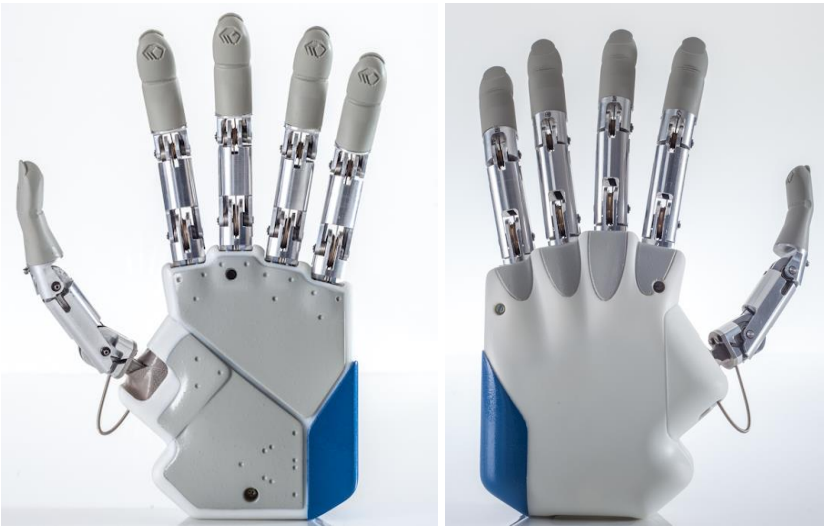

Fig. 1 Palm and dorsum view of IH2 Azzurra (left hand version).

As a safety feature the actuation of the five DoA is non-back-drivable. This means that motion is not transmitted from the fingers to the motors or in other words that in the case of failure of the power supply the hand will not reopen (hence let falling objects it might have held). The hand includes basic sensors for grasping, gesture and elementary manipulation:

- five motor encoders (one for each motor) that measure the posture of the fingers;
- five motor current sensors that measure the electrical current flowing into the motors (hence the grasping force);
- ten proximity sensors (two for each motor) that detect when the finger is fully flexed/abducted or extended/adducted;
- four tendon tension force sensors[1] (one for each flexion/extension action unit).

In addition to these, seven external sensors (analog) can be connected to the hand through one of the expansion ports. All sensors can be read, and all motors can be operated and controlled by means of the real-time Prensilia communication protocol described in chapters 4 and 5. In order to learn about all the features of *IH2 Azzurra* and hence its potential it is worth describing the internal mechanisms and principles of operation. Take some time to read the following paragraphs.

[1] Tendon tension force sensors are not available on all hands.

2.1 Mechanisms overview

The picture in Fig. 2A depicts the main components of the transmission train from the motor to the finger: the motor (in black and gray), two spur gears (in navy blue), the screw (in yellow), the linear slider (in green), the tendon (in red) and the finger. When the motor rotates the spur gears spin the screw which in turn moves the slider linearly and pulls/releases the tendon. The tendon runs into a Bowden cable (i.e. the tube used in bicycles breaks) from the exit of the actuation unit to the base of the finger and flexes/extends the finger. When the finger is completely extended or completely flexed the slider automatically stops. This behavior is achieved by means of proximity sensors at the two edges of the stroke of the slider (not shown in the picture).
Since the motor embeds an encoder it is possible to readout the position of the slider i.e. the amount of tendon released at any time. It is worth noting that the position of the slider does not correspond univocally to the posture of the finger; indeed the posture of the finger can adapt to the object, hence different postures of the finger can correspond to the same position of the slider. Once calibrated the position of the slider will range between 0 (finger completely extended) and 255 (finger completely flexed) if read through the communication protocol described in chapter 4.

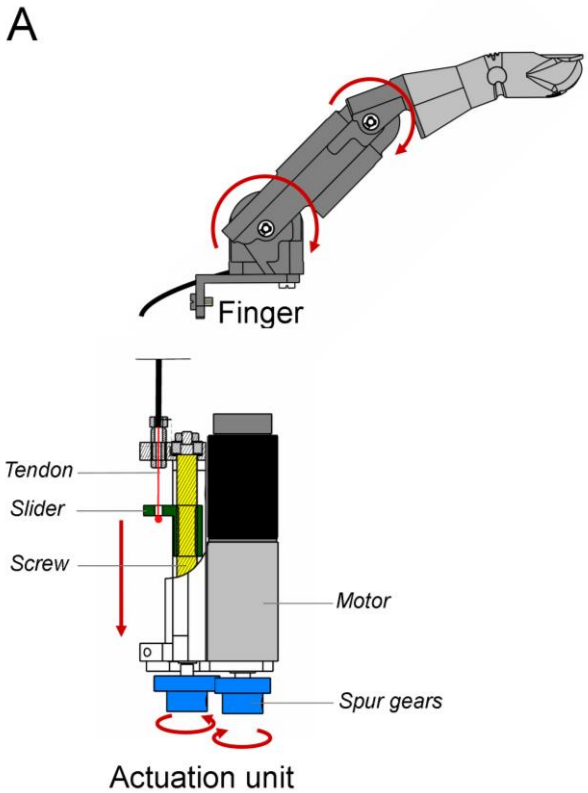

Fig. 2 Transmission of the flexion/extension DoA (A) and of the thumb abduction/adduction DoA (B).

Fig. 2B (NOT SHOWN) highlights the transmission of the thumb abduction/adduction DoA [..]. When the thumb is completely adducted or completely abducted it will automatically stop [..]. Like the others this motor embeds an encoder and it is possible to readout its position of the abduction/adduction at any time. Once calibrated the abduction/adduction position will range between 0 (finger completely adducted) and 255 (finger completely abducted).

2.2 Sensory system overview

IH2 Azzurra integrates: 5 motor rotary encoders for position measurement (one for each DoA), 5 motor current sensors, 4 tendon force tension sensors (one for each flexion/extension DoA)[2], and 10 proximity sensors (two for each DoA). Additionally, seven analog sensors can be added and acquired by the hand. Encoders and current sensors, and/or tendon tension sensors (if present) are used for position and force control during automatic grasps (cf. paragraph 2.3.2).

2.2.1 Motor rotary encoders

A rotary encoder, also called a shaft encoder, is an electro-mechanical device that converts the angular position or motion of a shaft (e.g. the shaft of a motor) to a digital code. There are two main types: absolute and incremental (relative) encoders. Incremental encoders are embedded in each of the five motors of *IH2 Azzurra*; these provide information about the motion of the motor shaft, which is processed by the internal controller into position information.
In *IH2 Azzurra* position information can be read in two ways: *raw* (or *incremental*) *position* and *calibrated* (or *absolute*) *position*. **Raw position** represents the reading from the incremental encoder without further processing; raw position is encoded in 17 bits. This means that the raw position could range between 0 and 131071 (unsigned). Raw position is relative to the start position (i.e. 0) which is the position of the motor/finger when the hand is switched ON. **Calibrated position** represents a processed information which is computed by the internal controller after the calibration procedure (cf. paragraphs 4.2.2 and 4.2.3). Calibrated position is encoded by 8 bits and ranges between 0 (finger completely extended / thumb completely adducted) and 255 (finger completely flexed / thumb completely abducted). The calibrated position resolution is hence around 0.78 deg. Commands for reading raw position or calibrated position are described in chapter 4.

2.2.2 Motor current sensors

Motor current, $I_{M,}$ is measured using current shunt monitors (Texas Instruments) and acquired by a 10 bit AD converter. The digital readings range between 0 and 1023 and respects the following law:

$$I_M\,|_{\text{flexion}} \approx I_M \cdot 901$$
$$I_M\,|_{\text{adduction}} \approx I_M \cdot 1230$$

For the flexion/extension or for the adduction/abduction DoA, respectively. The nominal motor current resolution is around 1.1mA for the flexion/extension DoA and 0.81 mA for the adduction/abduction DoA. Linearity is nominally achieved by the current shunt monitor in the range 10 and 1000 LSB (left hand) or 10 and 840 LSB (right hand). As an embedded safety feature, if the motor current overcomes the motor-specific **thermal limit current** (0.41 A for thumb abduction/adduction, 0.81 A for finger flexion/extension DoA) for longer than **1000 ms**, the motor is automatically switched off and put in STOP mode […].

2.2.3 Tendon tension sensors[2]

Tendon tension sensors are designed and produced by Prensilia S.r.l.; these are based on micro-machined cantilevers topped by semiconductor strain gauges that measure the force tension in the tendon (proportional to the grasping force of the fingers). For the ring-little actuation unit, the tendon tension sensor may be connected to one of the two fingers, by properly routing the tendon.

[2] Tendon tension force sensors are not available on all hands.

2.2.4 Proximity sensors

Hall effect digital proximity sensors (Allegro A3213 hall effect switches) are used to limit the physical range of the DoA and to automatically stop the motors. When they are active the DoA is either fully flexed (abducted) or extended (adducted).

2.2.5 External analog sensors

In addition to the sensors described above, **advanced users** could connect to the hand 7 additional external sensors, provided that their range is within 0-5 V. Details on how to connect external sensors to the hand can be found in the complete documentation.

2.2.6 Sensor data resolution and encoding table

The following table resumes the resolution of the different sensors and the data encoding.

Type	Binary encoding [#bits]	LSB Resolution
Raw position	17	-
Calibrated position	8	0.78 deg
Motor current (flexion)	10	1.1 mA
Motor current (abduction)	10	0.81 mA
Tendon tension force	10	-
External analog sensors	10	-

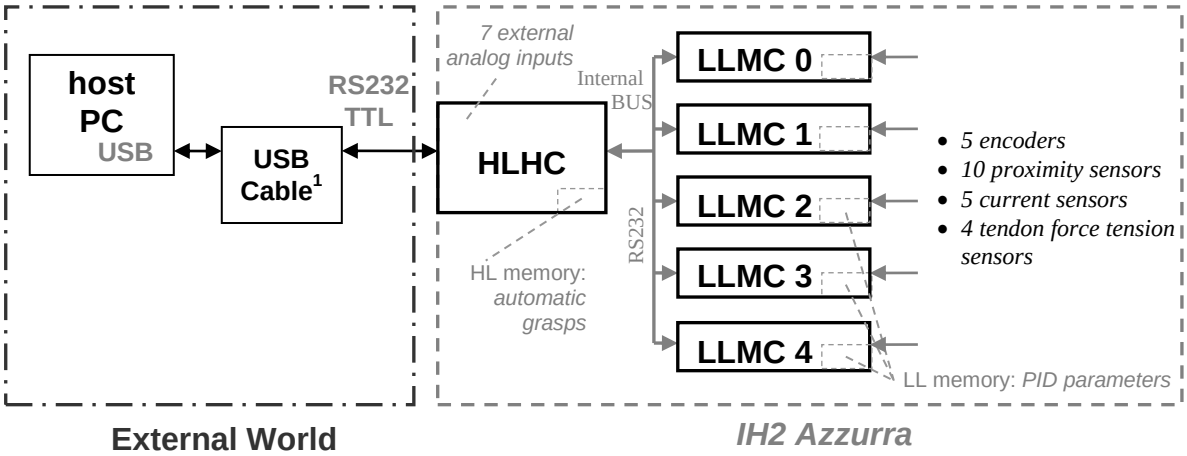

Fig. 3 *IH2 Azzurra* hierarchical control architecture[3].

2.3 Control system overview

The embedded controller in the hand is arranged in a hierarchical architecture consisting of 5 Low Level Motion Controllers (LLMC) and one High Level Hand Controller (HLHC) (Fig. 3). Each Degree of Actuation (DoA) is directly controlled by means of a LLMC that achieves position control, current control, hybrid (current/position) control, tendon tension force control[3], and monitors motor current absorption (ensuring a long-life operation of the motor). All LLMCs are controlled by the HLHC, that regulates overall hand operation, implements high level functions (like automatic grasps) and acts as interface with the external world. Although *IH2 Azzurra* comes with the USB cable provided (cf. chapter 3), it uses a standard serial communication bus (RS232,

[3] Tendon tension force sensors are not available on all hands.

TTL levels) to communicate with the external world and internally (HLHC-LLMC bus). *IH2 Azzurra* presents two layers of non-volatile memory (E^2PROM). The HLHC memorizes automatic grasp features (preshaping postures and force levels, cf. paragraph 4.4), whereas each LLMCs memorizes PID parameters (*Kp*, *Ki*, *Kd*) of the control algorithms.

The USB cable allows access to the sensors connected, to the functions implemented by the LLMCs and to those implemented by the HLHC, through the communication protocol described in this document (cf. chapters 4 and 5). <u>In particular by communicating with the LLMC you can (for each DoA/motor):</u>

- Read the raw position (17 bits) / motor-current / tendon tension force value[3];
- Set the speed / raw position (17 bits) / motor-current / tendon tension force[3] of one DoA to a desired value;
- Read / memorize (i.e. set in the internal memory) the PID parameters for the position / motor-current / tendon tension force[3] control loops;
- Read / memorize maximum motor-current or speed allowed by the DoA;
- Read the status of the DoA.

<u>By communicating with the HLHC you can (among the others):</u>

- Calibrate the hand;
- Read the calibrated position (8 bits) / motor-current / tendon tension force value[3] for a specific DoA;
- Set the speed / calibrated position (8 bits) / motor-current / tendon tension force[3] of one DoA at a desired value;
- Move the hand in a specific posture (i.e. move each DoA to a specific, desired position);
- Memorize automatic grasps;
- Launch automatic grasps.

⚠ The position of each DoA can be read in *raw* or in *calibrated* mode. The *raw* value (as it is read from the encoder, i.e. in a 17 bit data format) can be requested to the LLMC; the *calibrated* value (8 bit data format, range 0-255 meaning DoA closed-open, respectively) can be read from the HLHC.

2.3.1 Calibration routines

IH2 Azzurra features two calibration routines that can be issued by sending the appropriate commands over the communication bus; these are what is called the *first calibration* routine and *fast calibration* routine.

The *first calibration* should be issued once in a while and most important every time there is a change of the mechanics of the hand: this could be the case when a tendon is replaced, or the limits of the abduction/adduction DoA are manually moved. Indeed, the *first calibration* routine serves to map the open/close positions of the DoA to the position values of 0 and 255, respectively, and hence to readout consistent *calibrated position* values (as defined in section 2.2.1, page 8). Once issued every DoA fully closes and opens so that the raw position from the encoders is processed and memorized in the HLHC memory in order to map to the 0-255 range.

The *fast calibration*, as the name recalls, is a quicker routine: all DoA simply open and their position is mapped to 0 (zero). For correct operation of the HLHC functions of the hand (e.g. automatic grasps) the *fast calibration* should be the first command issued once the hand is switched ON. The *fast calibration* will work properly only if a *first calibration* was issued at any time before that moment. Although not strictly required the calibration routines can be issued at any time without causing problems.

2.3.2 Automatic grasps

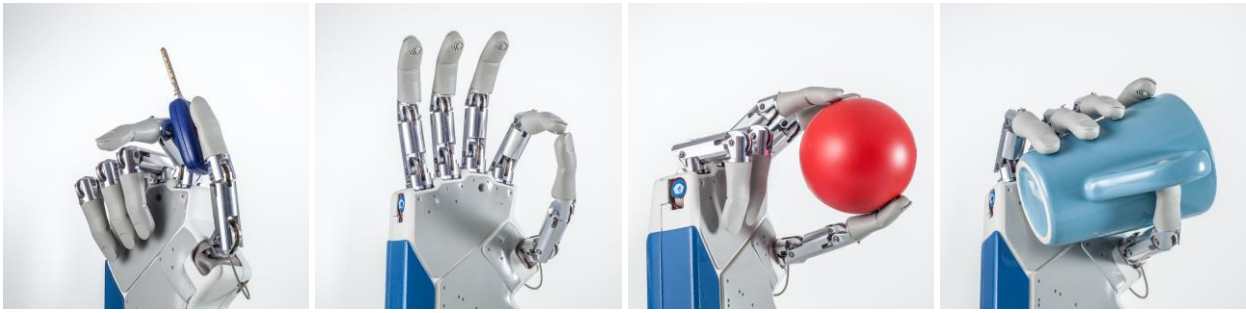

Fig. 4 Representative prototypical grasps. From left to right: lateral (or key grip), bi-digital (or pinch), tri-digital, power grasp.

IH2 Azzurra is able to perform grasps in an automatic, stereotypical (pre-programmed) manner. Grasps are modeled on natural grasping and triggered by simple commands received by the HLHC. In particular the original package includes ten prehension patterns (types of grasps, cf. Fig. 4) with user selectable force levels:

1. Lateral grasp (or key grip);
2. Thumb-index bi-digital (or pinch) grasp;
3. Thumb-index bi-digital (or pinch) grasp – last fingers extended;
4. Tri-digital grasp;
5. Tri-digital grasp – last fingers extended;
6. Cylindrical grasp;
7. Buffet grasp;
8. Ring/little grasp ("three" gesture);
9. Last three fingers grasp (pistol gesture);
10. Long fingers grasp (thumb up gesture).

The ten prehensile patterns differ in the number of digits involved in the enclosing phase of the grasp: e.g. in the lateral grasp i.e. the one used to hold a key or a credit card, only the thumb presses against the lateral aspect of the index finger, while all other fingers are passively positioned close to the palm. In the thumb-index grasp (the one used to pick up small objects) only the thumb and index are actively involved, whereas the last three fingers are kept apart. In the cylindrical grasp all fingers participate closing around the object, etc. The details about the number of digits involved are described in paragraph 4.3.7.

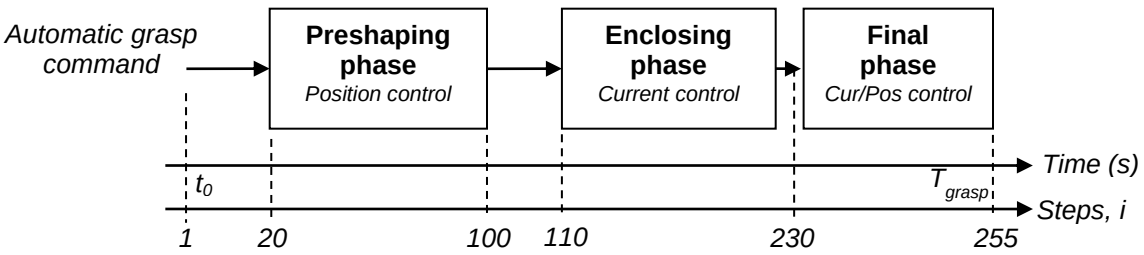

Fig. 5 Simplified schematic of control algorithm indicating two phases of control: selection of grasp by preshaping hand under position control, and grasping phase under force (motor current) control.

Automatic grasps as implemented by *IH2 Azzurra* are composed of three subsequent temporal phases (Fig. 5):

1. the *preshaping* phase – the digits open and are maintained;
2. the *enclosing* phase – the digits close;
3. the *final* phase – the digits stop.

The grasp is completed in T_{grasp} seconds and is divided in 255 temporal steps $i \in [1\ldots255]$; each phase kicks in at a specific time step and uses a different control mode (Fig. 5). This sequence is

shown in detail in Fig. 6 for one DoA. When an automatic grasp command is received the time step counter is started ($t=t_0$, $i=1$).

1. During the *preshaping* phase the DoAs (i.e. all motors) are first driven to match a pre-defined position (called *preshaping posture*), maintained between $i=1$ to $i=20$. This posture is functional to the prehensile pattern and can be preprogrammed by the user (cf. paragraphs 4.4.1 and following). The DoAs are then moved towards the *large aperture posture* (@ $i=100$), following a descending ramp using **position control laws** (Fig. 6). The *large aperture posture* is then maintained between $i=100$ to $i=110$.

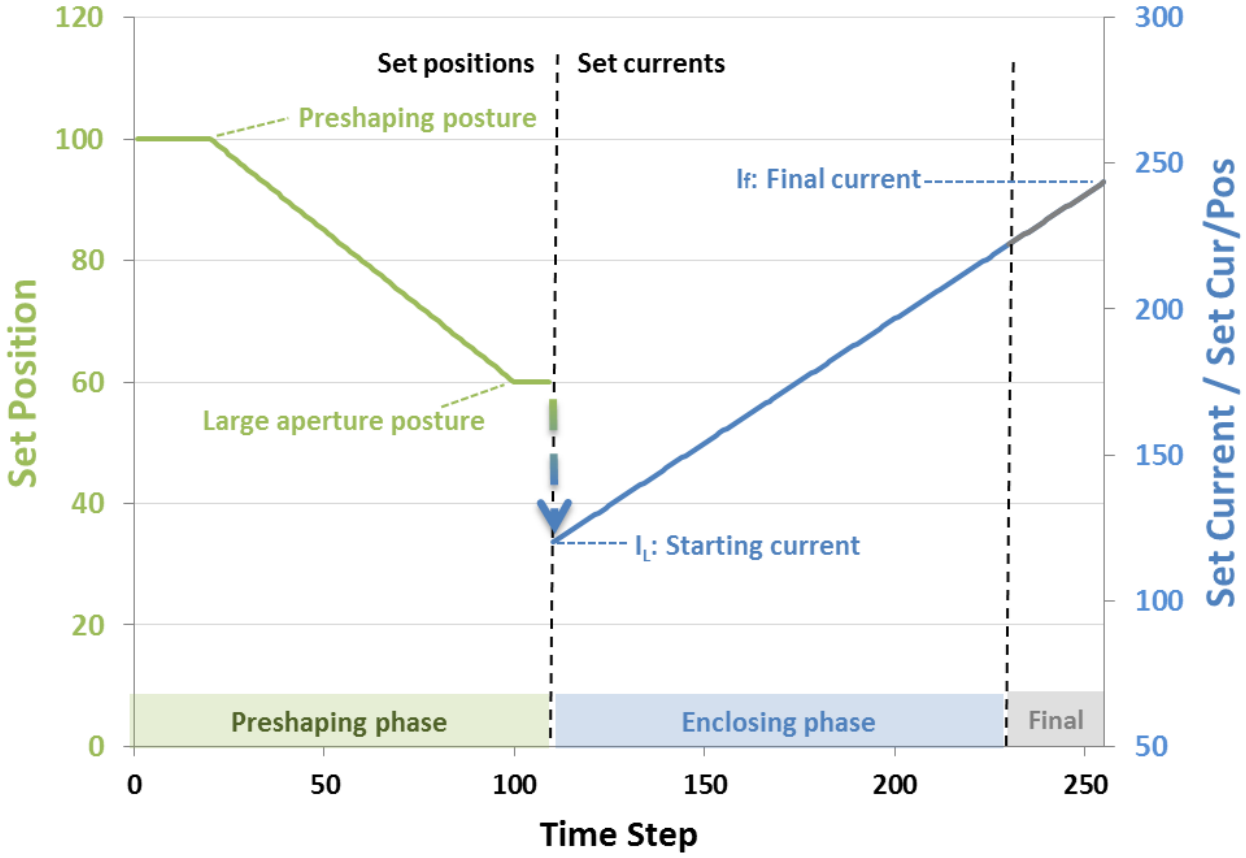

Fig. 6 Automatic grasp – Phases and timing during an automatic grasp.

2. In the *enclosing phase*, the hand closes the involved digits using **force control laws** (current control) ($i=110..230$).
3. The *final phase* kicks in at $i=230$ and ends at $i=255$ (corresponding to time $t=T_{grasp}$); during this time the involved fingers are closed using the hybrid current/position controller so that they stop when the desired force (current) is reached and the digits no longer move.

This motor scheme is bio-inspired by the human grasp, wherein grasp aperture is gradually molded as the hand reaches the target object.

During the *enclosing* and *final phases* the reference current follows a ramp (cf. Fig. 6) from a starting value I_L to a final value I_f, linearly ruled as:

$$I(i) = i\ \frac{I_f}{I_L} + I_L$$

I_L is a parameter which depend on each DoA; it is stored in the E^{2}PROM of the hand through the command MemLowCurr in paragraph 4.4.5 (pp. 34). I_f can be selected when issuing the automatic grasp command through the Grasp Force byte (cf. paragraph 4.3.7). Also the time to execute the grasp (T_{grasp}) is user selectable and programmable (cf. paragraph 4.4).

The graphs in Fig. 7 demonstrate the time course of the position and current readouts of the index finger during a recorded cylindrical grasp. In this example the *preshaping* and *large aperture postures* were 100 and 60, respectively, whereas I_L and I_f were 150 and 210. The grasp is executed in $T_{grasp}=2.5$ *s*. Fig. 8 shows the positions for the five DoAs during the same recorded grasp.

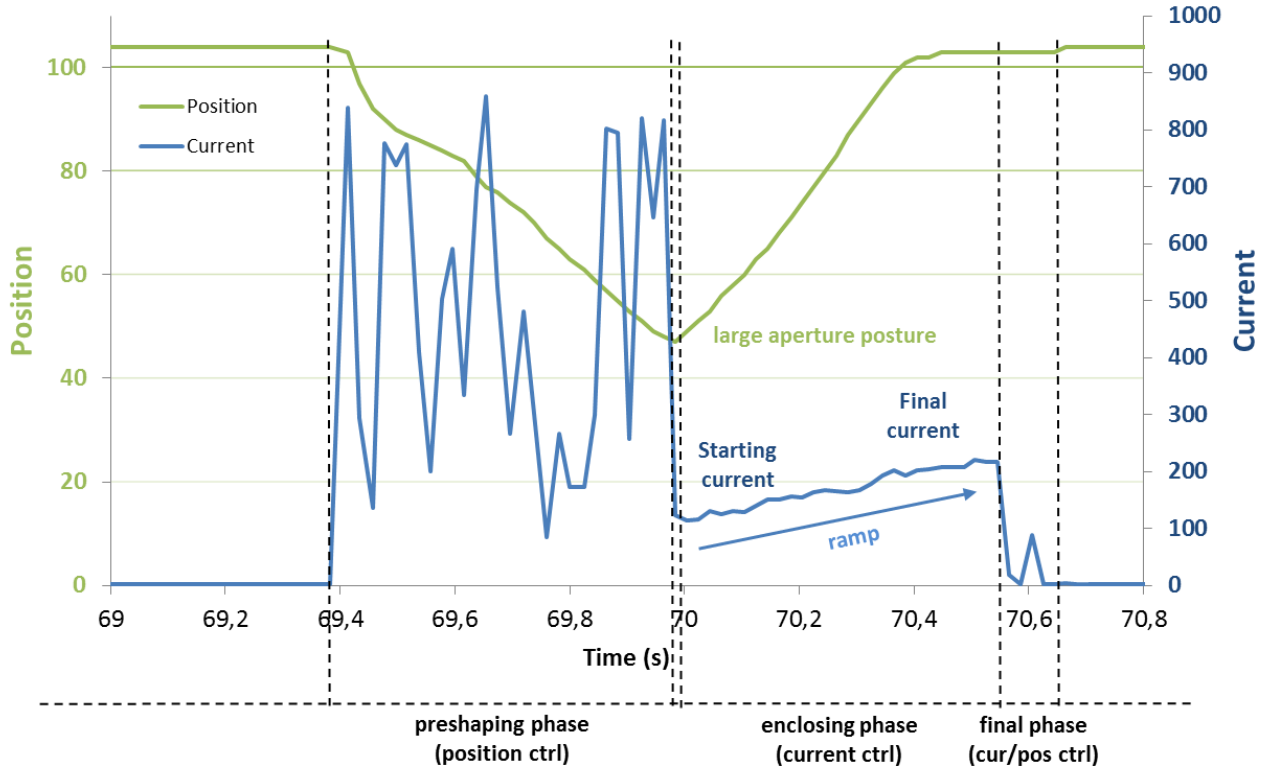

Fig. 7 Automatic grasp – index finger position and motor current vs. time during a cylindrical automatic grasp.

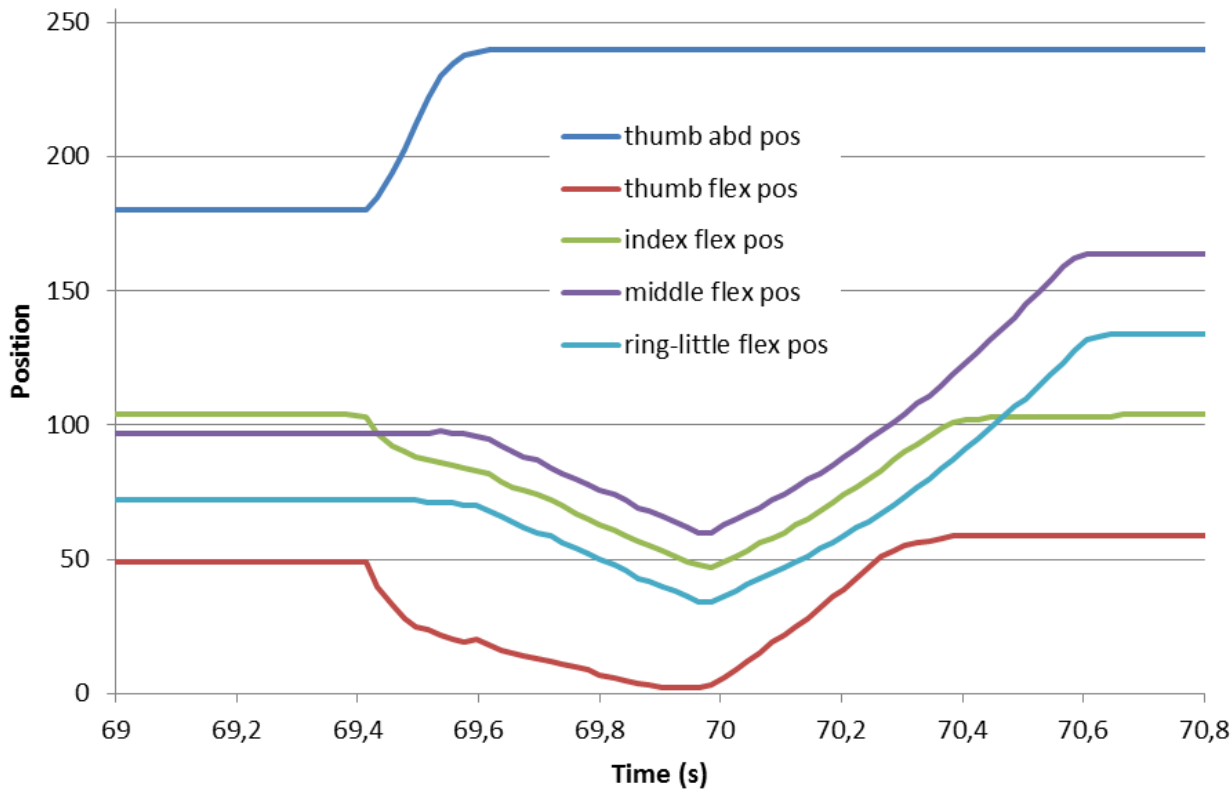

Fig. 8 Automatic grasp – hand posture vs. time with cylindrical automatic grasp.

⚠ Automatic grasps will not work if the hand is not calibrated.

2.4 Communication protocol overview

IH2-Azzurra receives commands from a host Personal Computer (PC) (or other controlling system) in the form of sequences of serial communication bytes called **packets**. The communication is achieved over a RS232 serial bus, which is accessible by means of the USB cable provided (see chapter 3). The HLHC in the hand buffers the incoming command stream and will only take an action once the entire packet has been received and the closing byte (where present) has been verified as correct. Non-existing packets will be ignored, whereas incomplete packets will make *IH2 Azzurra* wait for the completion of the packet. The command buffer will, however, be cleared whenever the hand is powered off/on.

2.4.1 RS232 (COM port) communication settings

The RS232 protocol settings required for communicating with *IH2 Azzurra* are listed in Table 1. The same settings are used in the internal communication bus between HLHC and LLMCs.

RS232 Asynchronous Communication (No handshake control).	
Serial wires used	TX, RX, GND
Baud Rate	115200 baud/sec
No. Bits	8
Stop bits	1
Parity	None
Voltage levels	TTL (0 +5V)

Table 1: LLMC communication protocol requirements.

2.4.2 Addresses of the motors

When communicating with *IH2 Azzurra*, each motor (or LLMC) is identified by a fixed Motor Address (MA), as follows:

Motor Address (MA)	DoA
0	Thumb abduction/adduction
1	Thumb flexion/extension
2	Index flexion/extension
3	Middle flexion/extension
4	Ring-little flexion/extension

An easy way to remember addresses is the following: thumb is the first finger, hence address is 1; index is the second finger (2), middle is the third (3), and ring is the fourth (4).

2.4.3 STATUS byte and control modes

Condensed information regarding the internal status of a specific LLMC can be obtained by requesting its **STATUS** byte, over the communication bus. Specifically the STATUS byte informs:

- **B7-B5:** about the type of control algorithm currently being implemented by the LLMC, which could be:
    a. STOP mode → the motor is still and not in control;
    b. PWM/Speed mode → the motor is driven with fixed PWM, i.e. in feed-forward speed control;
    c. Position mode → the motor is in position control;
    d. Tension mode → the flexion/extension motor (MA: 1÷4) is running in tendon force tension control;
    e. Current mode → the motor is **closing** with a specific motor current value (motor torque control);
    f. Current/Position mode → the motor is **closing** with a at specific motor current value (motor torque control) until it will stop moving; then it will break;
    g. COM ERROR → internal bus (HLHC-LLMCs) communication error;

- **B4:** if the desired control reference was reached or not;
- **B3:** if the open proximity sensor is ON (finger extended, thumb adducted);
- **B2:** if the close proximity sensor is ON (finger flexed, thumb abducted);
- **B1:** if the motor current overcame the maximum current allowed by the user or the thermal limit (paragraph 2.2.2, page 8);
- **B0:** if the motor is moving or not.

The data format of the STATUS byte is described below:

B7	**B6**	**B5**		**B4**	**B3**	**B2**	**B1**	**B0**
Control mode				**Ctrl OK**	**FCA**	**FCB**	**I**	**MOVING**
B7	B6	B5	Mode	1: OK 0: not OK	1: open	1: close	1: I>Imax 0: I<Imax	1: moving 0: steady
0	0	0	STOP					
0	0	1	PWM/Speed					
0	1	0	Position					
0	1	1	Tension (tendon)					
1	0	0	Current					
1	1	0	Current/Position					
1	1	1	COM ERROR					

Table 2 STATUS byte composition

Possible values for the STATUS byte are:
STATUS = 01010000 → Position control achieved.
STATUS = 01100100 → Tension control not achieved, finger closed.
STATUS = 00001000 → Stop, finger opened.
STATUS = 00000010 → Stop, current overflow.
STATUS = 11000001 → Current/Position control (motor still moving).
STATUS = 00010000 → Stop after current/position reached (motor stopped).
STATUS = 111xxxxx → Internal bus communication error (HLHC – LLMC bus).

An internal bus communication error might be due to improper logic levels or excessive noise on the bus due to high current spikes in the board (infrequent event, but the error is temporary and reversible), or due to unlike stopping of operation of the LLMC (very infrequent, and the error is irreversible).

2.4.4 Data binary encoding

Different data types are encoded using different numbers of bits. As an example while calibrated position is encoded by a single byte (8 bits, from 0 to 255, see also paragraph 2.2.1), motor current is encoded using 10 bits (from 0 to 1023). PWM/Speed data is encoded using 10 bits, 9 of which are used to code for speed (0-511) and 1 to select the direction (or sign) of rotation (0 DoA opens, 1 DoA closes). The following table, which resumes data encoding together with brief notes, should be kept in mind when communicating *IH2 Azzurra* using the communication protocol.

Data	Binary encoding [#bits]	Notes
PWM/Speed	10 (9 speed + 1 sign)	Sign = 0 DoA opens, sign = 0 DoA closes.
Raw position	17	Position relative to switch-ON posture of DoA.
Calibrated position	8	Works properly if hand calibrated.
Tendon tension force	10	*Sensor not available on all hands.*
Motor current	10	-

When a data type uses more than one byte the information has to be split and sent (or received) in two or more bytes. In this user guide we will refer to ByteU, ByteH and ByteL as *upper byte*, *high*

byte and *low byte*, respectively. The *low byte* is the byte that holds the least significant part of the data. If you think in terms of writing a bit pattern on paper, the *low byte* is the rightmost eight bits. The *high byte* holds the most significant part of the data in the case of data encoded by 2 bytes. It the data is encoded in 3 bytes is the upper byte to hold the most significant part. When communicating with the hand keep in mind that the most significant bytes will be sent/received first (**big-endian** format).

2.4.5 Read delays

Reading time delays are measured from the first raising edge of the transmission request, to the last falling edge of the receiver line (cf. Fig. 9). **Maximum time delay is 1 ms**.

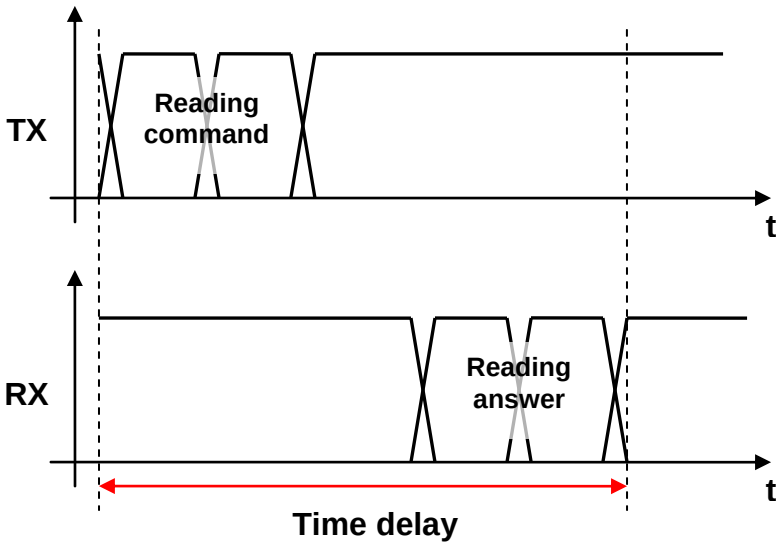

Fig. 9. Time delay measurement graph.

3 Installation

The prototype you purchased includes a (removable) wrist that can be used to mechanically connect the hand with external devices, like robot arms. The wrist provided is a threaded hole, sized M8, at the base of the hand. In the case of connection on a robot arm pay particular attention to the wiring from/to the hand.

A typical configuration for operating *IH2 Azzurra* hand is the one shown in Fig. 10, needing for:

- **A host PC** controlling the hand using the communication protocols detailed in this document;
- A bench power supplier with fixed regulated output (**9 V**) and a peak current of **5A** (not provided);
- A *TTL to USB Serial converter cable* (provided)[4] hereafter called USB cable.

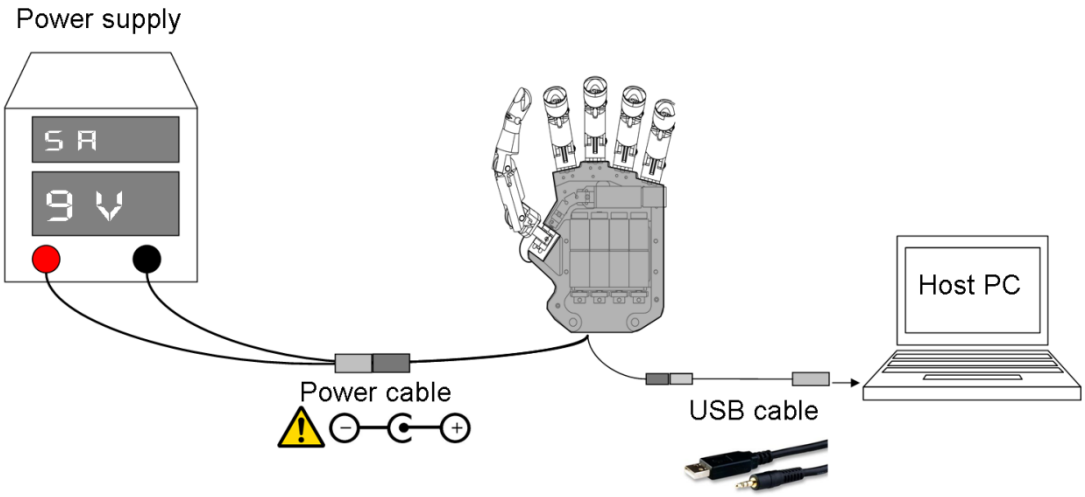

Fig. 10 Connection scheme between *IH2 Azzurra* hand, USB cable, host PC, and external power supply.

3.1 Install the USB cable

To operate the hand you first need to install the USB cable on your host PC. Drivers for the USB cable are freely available here[5] or from Prensilia; these are necessary to make the USB cable (and so *IH2 Azzurra*) appear as a virtual COM port. USB cable installation guides are available free here[6].

> ⚠ In some cases it might be useful to install the drivers manually, instead of having windows live update searching for the drivers automatically. If communication problems arise once the drivers are installed contact Prensilia at support@prensilia.com

The name of the COM port is specific to the operating system being used. The easiest way to find out which COM port is being assigned to *IH2 Azzurra* is to take note of what COM port appears when the USB cable is plugged in (provided the drivers have been installed on the PC already). In

[4] Manufacturer (FTDI) code: FTDI-TTL-232R-AJ. Spare parts available for purchase on Farnell (http://www.farnell.com/ code: 1740361) RS (http://www.rs-online.com/ code: 687-7774), or through Prensilia S.r.l.
[5] Third parties drivers: http://www.ftdichip.com/Drivers/VCP.htm
[6] Third parties installation guides: http://www.ftdichip.com/Support/Documents/InstallGuides.htm

the majority of cases (once the driver is installed), you can determine the COM Port on a MS Windows PC by following these steps:

1) Open Device Manager: Click: Start → Control Panel → Performance and Maintenance → System → Hardware → Device Manager
2) In the Device Manager list, look in Ports and find the COM Port, which was created by the USB driver. i.e.___ USB Serial Port___ (COM #).

3.2 Connect the cables and power-ON the hand

To operate *IH2 Azzurra* once the USB cable drivers are successfully installed

1) Connect the USB cable to the host PC;
2) Connect the USB cable to the 3.5mm stereo Jack connector on *IH2 Azzurra*;
3) Connect the black/red cable (provided) to an external power supply with regulated voltage output (+8V and GND), and to the power connector on *IH2 Azzurra*;
4) Switch-ON the power supply.
5) Once switched-ON the hand will start a ***greeting routine***: all DoA will open, then close and reopen again. During this routine the hand calibrates itself.
6) **The hand is now operative** and ready to communicate with the host PC!

⚠ The USB cable and the power supply cable can be connected in any order. However, if the USB cable is connected *after* the hand is switched-ON, some spurious signals could reach the hand through the serial port while physically connecting the plug. This could potentially issue a command or part of it and could leave in the host input queue spurious bytes as well, causing communication problems (if not correctly handled). It is suggested to switch-ON the hand after plugging in the USB connection.

⚠ Do not touch the metallic parts of the connectors (which are both connected to the system ground) once plugged in. The hand is a Static Sensitive Device!

4 HLHC commands

The High Level Hand Controller of *IH2 Azzurra* hand is accessible through the USB cable (provided) by means of variable length packets acting as recognized commands. Commands are divided into:

- **Single DoA commands**: for simple operation regarding one motor or sensor, such as finger movement, finger position control, reading of one sensor, etc.;
- **Status/Calibrate commands:** to obtain fingers/DoAs status and issue calibration routines;
- **Whole hand commands:** to automatically obtain complex operations related to the whole hand, such as automatic grasps;
- **Memory commands:** to store in the HLHC settings for the automatic grasps.

4.1 Single DoA commands

4.1.1 MoveMotor

Moves one motor at a desired speed and direction (**PWM/Speed mode**). The host PC sends 2 bytes; the hand sends back no bytes. The motor (or DoA) is selected by means of the nibble MA3..MA0 in the 1st byte of the packet. Direction is selectable by setting the Sign bit (S=0, finger/DoA opens; S=1 finger/DoA closes) in the 1st byte. Speed is selectable by setting the 9 bits D8..D0 (D8 is MSB, D0 is LSB) contained in the 1st and 2nd bytes.

Host PC transmits (TX)

Byte no.	1st								2nd							
Byte name	S+MA+MSD								LSD							
Bin	1	S	MA3	MA2	MA1	MA0	X	D8	D7	D6	D5	D4	D3	D2	D1	D0

Host PC receives (RX): No return message.

Example 1: abduct the thumb at full speed
Thumb abduction/adduction address is 0 → MA3..MA0 = 0000;
Abduct means closing the DoA → S = 1;
Full speed is 511 → D8..D0 = 111111111;
Result – host PC should transmit:
11000001 (or 11000011) followed by 11111111 → in hex 0xC1 (or 0xC3) followed by 0xFF.

Example 2: open middle at ¼ speed
Middle address is 3 → MA3..MA0 = 0011;
Open means → S = 0;
½ speed is 256 → D8..D0 = 100000000 → D8 = 1; D7..D0 = 00000000;
Result – host PC should transmit:
10001101 (or 10001111) followed by 00000000 → in hex 0x8D (or 0x8F) followed by 0x0.

4.1.2 SetFingerPosition

Moves one DoA towards a desired position (**position mode**) using *calibrated position* data format (8 bits: 0 DoA fully opened – 255 DoA fully closed). The host PC sends 3 bytes; the hand sends back no bytes. The motor (or DoA) is selected by means of the nibble MA3..MA0 in the 2nd byte of the packet. Once issued it is possible to know whether the position was reached or not by checking

B4 of the STATUS byte (**GetFingerStatus** command) or by simply reading the actual position (**GetFingerPosition** command).

TX

Byte no.	1st	2nd								3rd
Byte name	SFP	Motor Address								Position
Bin	01000100	0	0	0	0	MA3	MA2	MA1	MA0	00000000-11111111
Dec	68	0-4								0-255
Hex	0x44	0x0-0x4								0x0-0xFF

RX: No return message.

Example 3: place the index finger at half way
Index finger address is 2 → MA3..MA0 = 0010;
Half way means → Position = 128 → Position = 10000000;
Result – host PC should transmit:
01000100, 00000010, 10000000 → in hex 0x44, 0x2, 0x80.

Example 4: place the index and the ring-little fingers at half way
We can do this by sending two SetFingerPosition commands; host PC should transmit:
01000100, 00000010, 10000000, (for the index finger, as above, followed by)
01000100, 00000100, 10000000 (for the ring-little) → in hex: 0x44, 0x2, 0x80, 0x44, 0x4, 0x80.

4.1.3 SetFingerForce

Moves one DoA (all except for thumb abduction) in a closed loop fashion until desired tendon tension force is reached (**Tension mode**). The host PC sends 3 bytes; the hand sends back no bytes. The motor (or DoA) is selected by means of the nibble MA3..MA0 in the 2nd byte of the packet. The desired tendon tension force is selected by means of T9..T0 bits: T9, T8 (MSB_T) are contained in the 2nd byte; T7..T0 (LSB_T) are contained in the 3rd byte. Once issued it is possible to know whether the tendon tension force was reached or not by checking B4 of the STATUS byte (4.2.1 **GetFingerStatus** command) or by simply reading the tendon force sensor (4.1.7 **GetFingerForce** command).

SetFingerForce command is available only if tendon tension force sensors are installed.

TX

Byte no.	1st	2nd								3rd							
Byte name	SFF	MSB_T+MA								LSB_T							
Bin	01001010	T9	T8	x	x	MA3	MA2	MA1	MA0	T7	T6	T5	T4	T3	T2	T1	T0
Dec	74	MA + T8·64 + T9·128															
Hex	0x4A	MA + T8·0x40 + T9·0x80															

RX: No return message.

4.1.4 SetFingerCurrent

Closes one DoA with fixed motor current, i.e. in torque control (**Current mode**). The host PC sends 6 bytes; the hand sends back no bytes. The motor (or DoA) is selected by means of the 3rd (and 6th) byte of the packet. The desired motor current is selected by means of the 4th and 5th byte in a 10 bits data format (C9..C8 in the 4th byte, C7..C0 in the 5th byte). Once issued it is possible to

know whether the motor current was reached or not by checking B4 of the STATUS byte (4.2.1 **GetFingerStatus** command) or by simply reading the motor current sensor (4.1.8 **GetMotorCurrent** command).

TX

Byte no.	1st	2nd	3rd	4th	5th	6th
Byte name	*LLMC-C*	MA	SFC	MSB_C	LSB_C	MA
Bin	01011111		01100001	x..x C9, C8	C7..C0	
Dec	95	0-4	97			0-4
Hex	0x5F	0x0-0x4	0x61			0x0-0x4

RX: No return message.

⚠ **SetFingerCurrent** command works only in the closing direction.
⚠ The Motor Address (MA) should be the same in the 3rd and 6th byte of the packet otherwise the command will not be issued.
⚠ **SetFingerCurrent** command is actually a LLMC command (see chapter 5) and is reported also here for the sake of clarity.

4.1.5 SetFingerCurrPos

Closes one DoA with fixed motor current, i.e. in torque control and when the DoA stops moving (e.g. due to an object) the motor is stopped (**Current/Position mode**). The host PC sends 6 bytes; the hand sends back no bytes. The DoA is selected by means of the 2nd (and 6th) byte of the packet. The desired motor current is selected by means of the 4th and 5th byte in a 10 bits data format (CP9..CP8 in the 4th byte, CP7..CP0 in the 5th byte).

Once issued it is possible to know whether the motor current was reached or not, or whether the motor was stopped or not by checking the STATUS byte (cf. 4.2.1 **GetFingerStatus** command):

- if the DoA is moving but did not reach the target motor current value → STATUS = 1100 0000;
- if the DoA is moving and did reached the target motor current value → STATUS = 11010000;
- if the DoA was stopped as it reached the target motor current value and could no longer move → STATUS = 00010000.

TX

Byte no.	1st	2nd	3rd	4th	5th	6th
Byte name	*LLMC-C*	MA	SFCP	MSB_C	LSB_C	MA
Bin	01011111		01100110	x..x CP9, CP8	CP7..CP0	
Dec	95	0-4	102			0-4
Hex	0x5F	0x0-0x4	0x66			0x0-0x4

RX: No return message.

⚠
SetFingerCurrPos command works only in the closing direction.
⚠ The Motor Address (MA) should be the same in the 2nd and 6th byte of the packet otherwise the command will not be issued.
⚠
SetFingerCurrPos command is actually a LLMC command (see chapter 5) and is reported also here for the sake of clarity.

4.1.6 GetFingerPosition

Reads the *calibrated position* of one DoA. The host PC sends 2 bytes; the hand responds with 1 byte containing the *calibrated position*. The DoA is selected by means of the nibble MA3..MA0 in the 2nd byte of the packet.

TX

Byte no.	1st	2nd							
Byte name	GFP	Motor Address							
Bin	01000101	x	x	x	x	MA3	MA2	MA1	MA0
Dec	69	0-4							
Hex	0x45	0x0-0x4							

RX

Byte no.	1st
Byte name	Calibrated Position
Bin	00000000-11111111
Dec	0-255
Hex	0x0-0xFF

Example 5: plot the position of the thumb while moving

To make a plot, i.e. a time graph you will need to set a timer in your software environment. If you do so you can then issue **GetFingerPosition** commands at every timer loop and plot the position on your graph. In a descriptive manner the application on the host PC should:

1) issue a moving command to the thumb (e.g. close DoA at full speed): 0xC5, 0xFF;

2) start a timer, in which:

- **a**. issue **GetFingerPosition** command: 0x45, 0x1;
- **b**. wait for 1 byte;
- **c**. once returnd the 1 byte, plot the Position on the graph (or whatsoever indicator).

4.1.7 GetFingerForce

Reads the tendon tension force sensor of one DoA (all except for thumb abduction/adduction). The host PC sends 1 byte; the hand responds with 2 bytes containing the reading (T9..T0, 10 bits data format). The DoA is selected by means of the nibble MA3..MA0 in the byte; note that no tendon tension is associated to the thumb abduction/adduction motor (MA = 0). *Only available if tendon tension force sensors are available.*

TX

Byte no.	1st							
Byte name	GFF							
Bin	0	0	x	x	MA3	MA2	MA1	MA0
Dec	1-4							
Hex	0x1-0x4							

RX

Byte no.	1st								2nd							
Byte name	MSB_T								LSB_T							
Bin	x	x	x	x	x	x	T9	T8	T7	T6	T5	T4	T3	T2	T1	T0

GetFingerForce command is available only if tendon tension force sensors are installed.

4.1.8 GetMotorCurrent

Reads the motor current of one DoA. The host PC sends 2 bytes; the hand responds with 2 bytes containing the motor current (C9..C0, 10 bits data format). The DoA is selected by means of the nibble MA3..MA0 in the 2nd byte of the packet.

TX

Byte no.	1st	2nd							
Byte name	GMC	Motor Address							
Bin	01001001	x	x	x	x	MA3	MA2	MA1	MA0
Dec	73	0-4							
Hex	0x49	0x0-0x4							

RX

Byte no.	1st								2nd							
Byte name	MSB_C								LSB_C							
Bin	x	x	x	x	x	x	C9	C9	C7	C6	C5	C4	C3	C2	C1	C0

4.1.9 GetExternalSensor

Reads one of the external analog sensors connected to the main board. The host PC sends 2 bytes; the hand responds with 2 bytes containing the sensor value (S9..S0, 10 bits data format). The sensor is selected by means of the nibble B2..B0 in the 2nd byte of the packet.

TX

Byte no.	1st	2nd							
Byte name	GES	Address of external sensor							
Bin	01001101	x	x	x	x	x	B2	B1	B0
Dec	77	0-6							
Hex	0x4D	0x0-0x6							

RX

Byte no.	1st								2nd							
Byte name	MSB_S								LSB_S							
Bin	x	x	x	x	x	x	S9	S9	S7	S6	S5	S4	S3	S2	S1	S0

4.1.10 LowLevelVersion

Reads firmware version of the LLMC. The host PC sends 1 byte; the hand sends back 18 bytes containing the name of the firmware.

TX

Byte no.	1st	2nd	3rd	4th
Byte name	*LLMC-C*	MA	LLV	MA
Bin	01011111		01000000	
Dec	95	0-4	64	0-4
Hex	0x5F	0x0-0x4	0x66	

RX: ASCII string of 18 bytes (last byte is /NUL char)

4.2 Status/Calibrate commands

4.2.1 GetFingerStatus

Check DoA STATUS byte. The host PC sends 2 bytes; the hand responds with 1 byte containing the STATUS. The DoA is selected by means of the nibble MA3..MA0 in the 2nd byte of the packet (see also paragraph 2.4.3).

TX

Byte no.	1st	2nd							
Byte name	GFS	Motor Address							
Bin	01001011	x	x	x	x	MA3	MA2	MA1	MA0
Dec	75	0-4							
Hex	0x4B	0x0-0x4							

RX: Status Byte

B7	B6	B5		B4	B3	B2	B1	B0
Control mode				Ctrl OK	FCA	FCB	I	MOVING
B7	B6	B5	Mode	1: OK 0: not OK	1: open	1: close	1: I>Imax 0: I<Imax	1: moving 0: steady
0	0	0	STOP					
0	0	1	PWM/Speed					
0	1	0	Position					
0	1	1	Tension (tendon)					
1	0	0	Current					
1	1	0	Current/Position					
1	1	1	COM ERROR					

Example 6: execute a position trajectory with one finger

To track a sequence of desired positions it might be important to be sure that each position was correctly reached before moving to the next one. To do so the STATUS byte can be used in a very efficient manner. Indeed bit B4 of the byte acknowledges the host controller if the controller reached the target. If e.g. the finger to move is the middle the application on the host PC should:

1) issue a SetFingerPosition to position p[*n*] of the trajectory → in hex 0x44, 0x3, p[*n*];

2) in a timely fashion (e.g. every 10 ms):

- **a**. issue GetFingerStatus command: 0x4B, 0x3;
- **b**. wait for the STATUS byte;
- **c**. check if B4 → in c language: (STATUS>>4)&0x1
- **d**. if B4 = 1 the position is reached, hence set next position (*n*=*n*+1) and go to 1);
- **e**. if B4 = 0 the position is not reached, hence go to **a**.

4.2.2 FirstCalibration

This command is **mandatory**, every time mechanical changes occur to the DoAs of *IH2 Azzurra* (e.g. tendon substitution, proximity sensors replacement, different tendon pre-tensioning, etc). The host PC sends 1 byte; the hand sends back no bytes. FirstCalibration command starts a calibration routine and stores in the internal memory of the HLHC correct references, fundamental for *calibrated position* control (4.1.2 SetFingerPosition) and readout. Once sent to the HLHC, each motor completely opens, completely closes, and re-opens again (no obstacles should interfere

during the calibration routine). During this routine IH2 Azzurra will not process incoming commands.

TX

Byte no.	1st
Byte name	FiC
Bin	01000010
Dec	66
Hex	0x42

RX: No return message.

⚠ The FirstCalibration routine should be executed without objects or any other obstacle interfering with the motion of the hand! During the calibration routine no other commands should be issued as they will be discarded.

4.2.3 FastCalibration

This must be the first command to send when the hand is switched-ON. It is necessary for calibrated position control, as it retrieves correct references from the internal memory (stored during last FirstCalibration command). Once the command is issued, all fingers completely open, and during that time no other commands should be sent.

TX

Byte no.	1st
Byte name	FaC
Bin	01000110
Dec	70
Hex	0x46

RX: No return message.

⚠ The calibration routine should be executed without objects or any other obstacle interfering with the motion of the hand! During the calibration routine no other commands should be issued as they will be discarded.

4.3 Whole hand commands

4.3.1 HighLevelVersion

Reads firmware version of the HLHC. The host PC sends 1 byte; the hand sends back 18 bytes containing the name of the firmware.

TX

Byte no.	1st
Byte name	SA
Bin	0111 0010
Dec	114
Hex	0x72

RX: ASCII string of 18 bytes (last byte is /NUL char)

4.3.2 StopALL

IH2 Azzurra will stop all movements and ongoing control loops, once this command is issued: each DoA will then go in STOP mode.

TX

Byte no.	1st
Byte name	SA
Bin	01000001
Dec	65
Hex	0x41

RX: No return message.

4.3.3 SetHandPosture

With this command all motors of the hand move towards a desired position, hence posturing the hand. The host PC sends 7 bytes; the hand sends back no bytes. The first and last byte of the packet should be the same otherwise the command will not be issued. 2nd to 6th byte contain the 5 target positions in a ordered manner, starting from MA=0 (thumb abduction/adduction) and ending with MA=4 (ring-little flexion/extension). The reaching of the pre-shaping position may be monitored by sending multiple **GetFingerPosition** or **GetFingerStatus** commands.

TX

Byte no.	1st	2nd	3rd	4th	5th	6th	7th
Byte name	PSH	Target position 0	Target position 1	Target position 2	Target position 3	Target position 4	PSH
Bin	01001000	0..0-1..1	0..0-1..1	0..0-1..1	0..0-1..1	0..0-1..1	01001000
Dec	72	0-255	0-255	0-255	0-255	0-255	72
Hex	0x48	0x0-0xFF	0x0-0xFF	0x0-0xFF	0x0-0xFF	0x0-0xFF	0x48

RX: No return message.

⚠ The HLHC takes around 3 ms to implement the command, hence for a proper synchronization and real time (non-delayed) operation, new **SetHandPosture** commands should not be sent faster than every **3 ms**.

4.3.4 OpenALL

IH2 Azzurra will reopen flexion/extension DoAs (i.e. not the thumb abduction/adduction).

TX

Byte no.	1st
Byte name	OA
Bin	01001100
Dec	76
Hex	0x4C

RX: No return message.

4.3.5 EnableStreaming

With this command the *streaming mode* is enabled. The hand sends continuously and with defined time rate (*Ts*) packets of data (with defined packet size, *Ps*) pertaining the sensor readouts as configured by the parameter **StreamMode**. StreamMode can assume one of the following values:

- **1**: to receive packets containing current and position data. (*Ts=15 ms*, *Ps*= 20 bytes)
- **2**: to receive packets containing current data. (*Ts=10 ms*, *Ps* = 15 bytes)
- **3**: to receive packets containing position data. (*Ts=10 ms*, *Ps* = 10 bytes)
- **4**: to receive packets containing 7 external analog sensors data. (*Ts=5 ms*, *Ps* = 19 bytes)
- **5**: to receive packets containing tendon tension data. (*Ts=10 ms*, *Ps* = 15 bytes)
- **6**: to receive packets containing tendon tension and position data. (*Ts=15 ms*, *Ps* = 20 bytes)
- **7**: to receive packets containing positions and 7 ext. sensors (*Ts=15 ms*, *Ps* = 24 bytes)

TX

Byte no.	1st	2nd
Byte name	ESM	StreamMode
Bin	01000011	00000001-00000111
Dec	67	1-7
Hex	0x43	0x1-0x7

RX: Stream packets:

StreamMode 1: current and position

HeaderH	HeaderL	Size	Thumb ab/ad			Thumb			Index			Middle			Ring-little			EndH	EndL
0xAA	0x55	20	C_H	C_L	P	C_H	C_L	P	C_H	C_L	P	C_H	C_L	P	C_H	C_L	P	0x55	0xAA

StreamMode 2: current

HeaderH	HeaderL	Size	Thumb ab/ad		Thumb		Index		Middle		Ring-little		EndH	EndL
0xAA	0x55	15	C_H	C_L	C_H	C_L	C_H	C_L	C_H	C_L	C_H	C_L	0x55	0xAA

StreamMode 3: position

HeaderH	HeaderL	Size	Thumb ab/ad	Thumb	Index	Middle	Ring-little	EndH	EndL
0xAA	0x55	10	Position	Position	Position	Position	Position	0x55	0xAA

StreamMode 4: external analog sensors

HeaderH	HeaderL	Size	AN0		AN1		..	AN6		EndH	EndL
0xAA	0x55	19	MSB	LSB	MSB	LSB	..	MSB	LSB	0x55	0xAA

StreamMode 5: tendon tension

HeaderH	HeaderL	Size	Thumb ab/ad		Thumb		Index		Middle		Ring-little		EndH	EndL
0xAA	0x55	15	x	x	T_H	T_L	T_H	T_L	T_H	T_L	T_H	T_L	0x55	0xAA

StreamMode 6: tendon tension and position

HeaderH	HeaderL	Size	Thumb ab/ad			Thumb			Index			Middle			Ring-little			EndH	EndL
0xAA	0x55	20	x	x	P	T_H	T_L	P	T_H	T_L	P	T_H	T_L	P	T_H	T_L	P	0x55	0xAA

StreamMode 7: position and external analog sensors

HeaderH	HeaderL	Size	Thumb ab/ad	..	Ring-little	AN0		AN1		..	AN6		EndH	EndL
0xAA	0x55	24	Position	Position	Position	MSB	LSB	MSB	LSB	..	MSB	LSB	0x55	0xAA

⚠ If the streaming is enabled reading commands should not be issued, as the answer of the hand might be interleafed with the streaming data **synching it might be not trivial**.

4.3.6 DisableStreaming

With this command the *streaming mode* is disabled.

TX

Byte no.	1st
Byte name	DSM
Bin	01000111
Dec	71
Hex	0x47

RX: No return message.

4.3.7 Automatic Grasps

When this command is issued *IH2* performs an automatic grasp in a stereotypical, pre-programmed manner following the scheme presented in paragraph 2.3.2. The hand receives 4 bytes including the command code (0x6F), the grasp type (GT), grasp force (GF), and grasp duration (GD).
The GT indicates the prehensile grasping form or motor primitive (cylindrical grasp, lateral grasp, etc.); the GF indicates the grip force degree ranging from a very light grasp to a very powerful grasp. The GD represents the duration of the grasp. Refer to the following paragraphs for detailed description.

TX

Byte no.	1st	2nd	3rd	4th
Byte name	AG	Grasp Type	Grasp Force	Grasp Duration
Bin	0110 1111	0..0-1..1	0..0-1..1	>1110
Dec	111	0-255	0-255	>14
Hex	0x6F	0x0-0xFF	0x0-0xFF	>0xE

RX: No return message.

GRASP TYPE - GT

The *IH2 Azzurra* hand is capable of performing ten possible grasps, which can be issued using the GT byte. In the practice, the hand will assume a specific *preshaping posture* and *large aperture posture* based on the GT code received with this command. The list of possible grasps with their relative GT code is listed in the following table. The commands required to memorize the *preshaping postures* relative to the specific grasps is also listed. The procedure to memorize these postures in the non-volatile memory of the hand are described in paragraphs 4.4.1 - 4.4.4. As depicted from the table, when the *preshaping postures* for first four grasps are correctly memorized, the other automatic grasps do not require any further operation.

Grasp Type code (decimal)	Full name	DoAs involved in enclosing/final phases	Memorize preshaping posture using:
4	Cylindrical grasp	1, 2, 3, 4, 5	MemCylPreShape: Cyl[0]…Cyl[4]
3	Tri-digit grasp	1, 2, 3	MemTriPreShape: Tri[0]...Tri[4]
2	Bi-digit grasp	1, 2	MemBiPreShape: Bi[0]…Bi[4]
1	Lateral grasp	1	MemLatPreShape: Lat[0]…Lat[4]
31	Tri-digit last digits extended	1, 2, 3	*Automatic*
21	Bi-digit last digits extended	1, 2	*Automatic*
11	Buffet grasp	1	*Automatic*
6	"Three" grasp	5, 4	*Automatic*
7	"Pistol" grasp	5, 4, 3	*Automatic*
8	"thumb up" grasp	5, 4, 3, 2	*Automatic*
0	Relax	1, 2, 3, 4, 5	*Automatic*

GRASP FORCE - GF

The graph in Fig. 11 recall the motor scheme implemented by the automatic grasps. We are interested in the *enclosing* phase now. While the starting current, I_L, depends on the friction of the mechanical transmission of a specific DoA, the final current, I_f, is selectable by the user when issuing an automatic grasp command. In particular the I_f depends on the GF as ruled by the following equation:

$$I_f = GF \ \frac{3\, I_H}{255} + I_L$$

Where I_H is a parameter that varies across DoAs and depends on the friction of the mechanical transmission. Since both I_L and I_H are stored in the non-volatile memory of the hand, the GF byte univocally determines the final current for each motor during a grasp. Consequentially, the GF which is sent at the time of issuing an automatic grasp command, determines the grip force. The procedures on how to store I_L and I_H in the non-volatile memory of the hand are described in paragraphs 4.4.5 (MemLowCurr command) and 4.4.6 (MemHighCurr command), respectively.

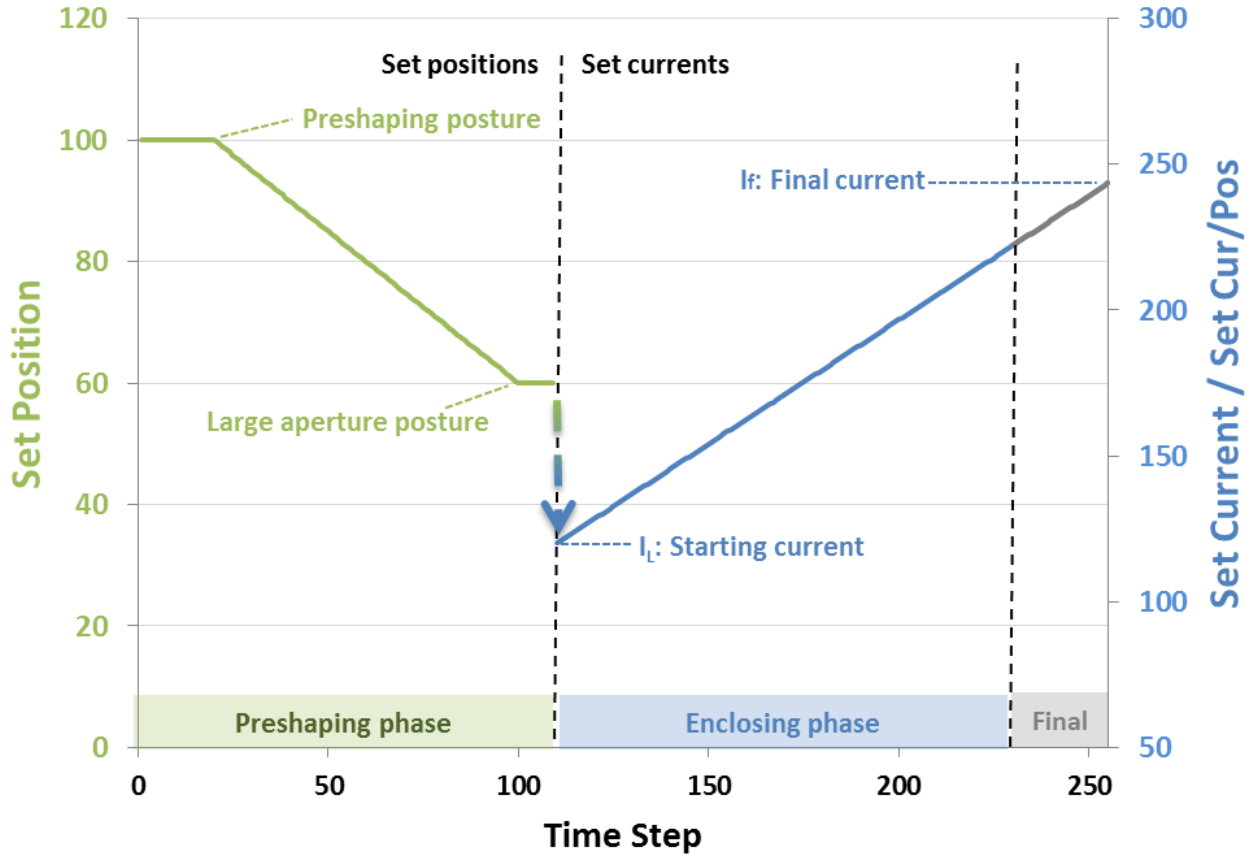

Fig. 11 Automatic grasp – Phases and timing during an automatic grasp.

GRASP DURATION - GD

The GD byte determines the maximum time duration of the grasp in multiples of **52 ms**. In particular GD the time when the last current reference is set. In turn the maximum grasp duration can be computed as:

$$T_{grasp} = (\text{GD-1}) \cdot 52$$

For example, with GD=25 the grasp will last at most 24·52 ms ≈ 1.25 seconds; with GD 255 (upper limit) the grasp will last ≈ 13.00 seconds. There are some considerations that should be considered when choosing the optimal GD:

- **GD > 15** – If a GD ≤ 15 is sent, a GD of 15 will be used (15·52 ms = 0.78 seconds). This constrain is due to the maximum speed of the motors deployed in the hand which would not allow such fast grips. A GD=15 should be deemed as a limit and might not operate properly. A GD ≥ 25 is warmly suggested by Prensilia for your *IH2 Azzurra* hand prototype.

- **GD and GF should be chosen as a pair** – An automatic grasp will operate properly if the GD and the GF do not contrast with the physical principles of the system. Since the GF regulates the final current (cf. equation above), larger GF values correspond to faster digit movements, during the enclosing phase. The GD instead, determines the time $t=T_{grasp}$ corresponding to the 255th time step in which the grasp is divided or in other words, the time when the last current reference is set. In turn if the GF is *too large* with respect to the GD value, the automatic grasp will *practically* end earlier than the time imposed by the GD (GD·52 ms).

4.3.8 Grasp Stepper

When this command is issued *IH2* "jumps in" a certain step of the selected automatic grasp.

TX

Byte no.	1st	2nd	3rd	4th	54th
Byte name	Header	Grasp Step	Grasp Type	Grasp Force	Header
Bin	0110 1111	0..0-1..1	0..0-1..1	0..0-1..1	0110 1111
Dec	111	0-255	0-255	0-255	111
Hex	0x4E	0x0-0xFF	0x0-0xFF	0x0-0xFF	0x4E

RX: No return message.

Different control modes are used depending on the step provided:

- 25 < Step < 110: IH2 opens following the predefined trajectory of the automatic grasp (Position Control);
- 110 < Step < 230: IH2 closes using Current Control;
- 230 < Step < 255: IH2 closes using Current/Position Control;

4.4 HLHC memory commands

⚠ HLHC memory commands change the internal settings used for grasps! New stored settings are maintained after power off. Pay special attention when adjusting these values!

Preshaping posture memorization commands

The following commands are used to store in the HLHC non-volatile memory the *preshaping postures* for the different automatic grasps. They all follow the same data packet format. The host PC sends 7 bytes; the hand sends back no bytes. The first and last byte of the packet should be the same otherwise the command will not be issued. 2nd to 6th byte contain the 5 positions in a ordered manner, starting from MA=0 (thumb abduction/adduction) and ending with MA=4 (ring-little flexion/extension) similar to the SetHandPosture command. The procedure to memorize a *preshaping posture* is very simple. The user only needs to identify the position of each DoA so that the hand mimics that specific grasp, when the digits are not closed yet. This can be done by moving the digits in the positions illustrated in in Fig. 12, reading such positions and memorizing them using the following preshaping commands.

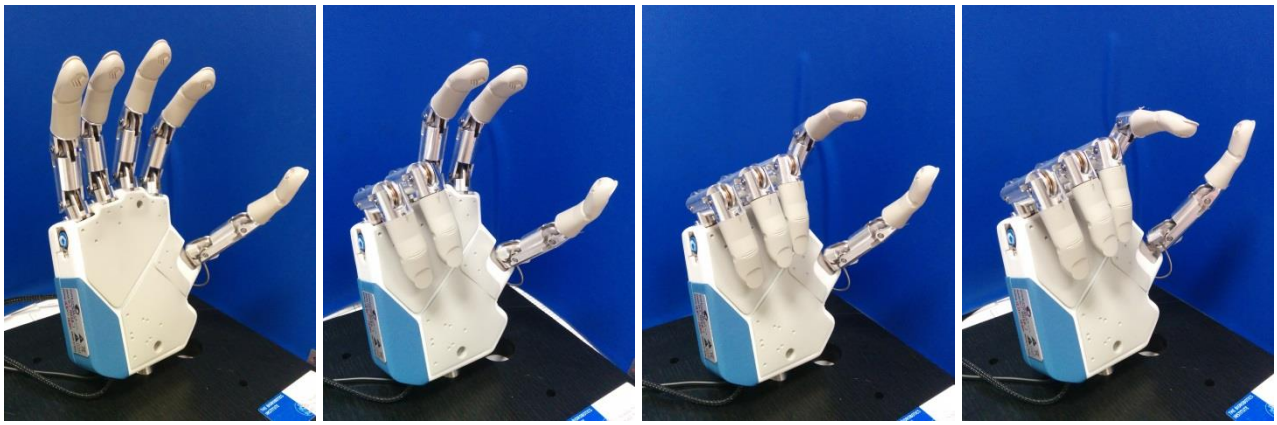

Fig. 12 – Hand morphed in the preshaping postures for the cylindrical, tri-digit, bi-digit and lateral grasps.

4.4.1 MemCylPreShape

Stores in the HLHC non-volatile memory the *preshaping posture* for the cylindrical grasp.

TX

Byte no.	1st	2nd	3nd	4th	5th	6th	7th
Byte name	MCPS	Preshape posture 0	Preshape posture 1	Preshape posture 2	Preshape posture 3	Preshape posture 4	MCPS
Bin	01001000	0..0-1..1	0..0-1..1	0..0-1..1	0..0-1..1	0..0-1..1	01001000
Dec	88	0-255	0-255	0-255	0-255	0-255	88
Hex	0x58	0x0-0xFF	0x0-0xFF	0x0-0xFF	0x0-0xFF	0x0-0xFF	0x58

RX: No return message.

4.4.2 MemTriPreShape

Stores in the HLHC non-volatile memory the *preshaping posture* for the tri-digit grasp.

TX

Byte no.	1st	2nd	3nd	4th	5th	6th	7th
Byte name	MTPS	Preshape posture 0	Preshape posture 1	Preshape posture 2	Preshape posture 3	Preshape posture 4	MTPS
Bin	01001000	0..0-1..1	0..0-1..1	0..0-1..1	0..0-1..1	0..0-1..1	01001000
Dec	90	0-255	0-255	0-255	0-255	0-255	90
Hex	0x5A	0x0-0xFF	0x0-0xFF	0x0-0xFF	0x0-0xFF	0x0-0xFF	0x5A

RX: No return message.

4.4.3 MemBiPreShape

Stores in the HLHC non-volatile memory the *preshaping posture* for the b-digit grasp.

TX

Byte no.	1st	2nd	3nd	4th	5th	6th	7th
Byte name	MBPS	Set Position0	Set Position1	Set Position2	Set Position3	Set Position4	MBPS
Bin	01001000	0..0-1..1	0..0-1..1	0..0-1..1	0..0-1..1	0..0-1..1	01001000
Dec	91	0-255	0-255	0-255	0-255	0-255	91
Hex	0x5B	0x0-0xFF	0x0-0xFF	0x0-0xFF	0x0-0xFF	0x0-0xFF	0x5B

RX: No return message.

4.4.4 MemLatPreShape

Stores in the HLHC non-volatile memory the *preshaping posture* for the lateral grasp.

TX

Byte no.	1st	2nd	3nd	4th	5th	6th	7th
Byte name	MLPS	Set Position0	Set Position1	Set Position2	Set Position3	Set Position4	MLPS
Bin	01001000	0..0-1..1	0..0-1..1	0..0-1..1	0..0-1..1	0..0-1..1	01001000
Dec	89	0-255	0-255	0-255	0-255	0-255	89
Hex	0x59	0x0-0xFF	0x0-0xFF	0x0-0xFF	0x0-0xFF	0x0-0xFF	0x59

RX: No return message.

Force levels memorization commands

The following commands are used to store in the HLHC non-volatile memory the three force levels (low, medium and high) used in the different automatic grasps. They all follow similar data packet format. The host PC sends 10 (6 in Tension mode) bytes; the hand sends back no bytes. The first and last byte of the packet should be the same otherwise the command will not be issued.
In the case of enclosing phase using **Current/Position** mode the 2nd to 9th byte in the packet contain the 4 motor current target values (10 bits format, 0-1023) in a ordered manner, starting from MA=1 (thumb flexion/extension) and ending with MA=4 (ring-little flexion/extension). CurrH*n* and CurrL*n* refer to the most (H) significant byte and least (L) significant byte, respectively.

In the case of enclosing phase using **Tension** mode the 2nd to 5th byte in the packet contain the 4 tendon tension force deltas in a ordered manner, starting from MA=1 (thumb flexion/extension) and ending with MA=4 (ring-little flexion/extension).

4.4.5 MemLowCurr - I_L

Stores the motor current value for the low force grasp, in Current/Position mode grasps.

TX

	1st	2nd	3nd	4th	5th	6th	7th	8th	9th	10th
Name	MLC	CurrH1	CurrL1	CurrH2	CurrL2	CurrH3	CurrL3	CurrH4	CurrL4	MLC
Bin	01101100	x00-x11	0..0-1..1	x00-x11	0..0-1..1	x00-x11	0..0-1..1	x00-x11	0..0-1..1	01101100
Dec	110	0-3	0-255	0-3	0-255	0-3	0-255	0-3	0-255	110
Hex	0x6E	0x0-0x3	0x0-0xFF	0x0-0x3	0x0-0xFF	0x0-0x3	0x0-0xFF	0x0-0x3	0x0-0xFF	0x6E

RX: No return message.

The optimal I_L should be chosen through an iterative procedure. For the thumb, index, and middle flexion/extension (MA=1 to MA=3), I_L should be equal to the minimum current needed to close the finger to a position >200. Thus, the user should:

1. Fully extend the finger;
2. Set the motor current to C=50 by issuing a SetFingerCurrent command;
3. Wait for the finger to stop;
4. Read the HLHC position reached by the finger by issuing a GetFingerPosition command;
5. If the position received is <200, repeat from (1) after increasing C by 10, otherwise set I_L as C

For MA=4 (ring-little flexion/extension) I_L should be set in order to match the other fingers speed while performing a cylindrical automatic grasp. Fingers speed can be compared both visually and with the help of the Streaming Panel of the Demo Application (cf. Chapter 6), as shown in Fig. 13.

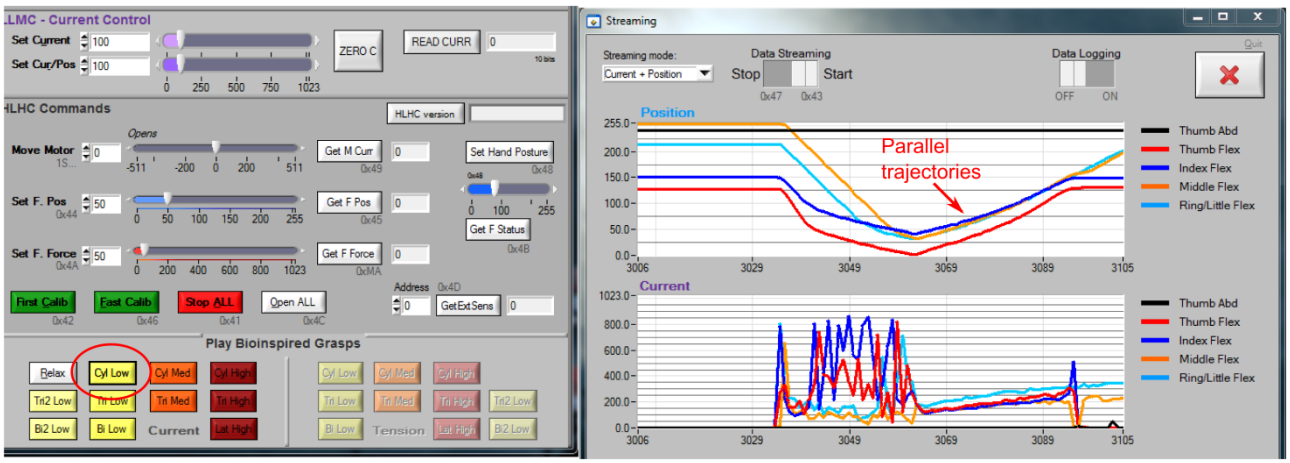

Fig. 13 Automatic grasp – Phases and timing during an automatic grasp.

4.4.6 MemHighCurr - I_H

Stores the motor current value for the high force grasp, in Current/Position mode grasps.

TX

	1st	2nd	3nd	4th	5th	6th	7th	8th	9th	10th
Name	MHC	CurrH1	CurrL1	CurrH2	CurrL2	CurrH3	CurrL3	CurrH4	CurrL4	MHC
Bin	01101100	x00-x11	0..0-1..1	x00-x11	0..0-1..1	x00-x11	0..0-1..1	x00-x11	0..0-1..1	01101100
Dec	108	0-3	0-255	0-3	0-255	0-3	0-255	0-3	0-255	108
Hex	0x6C	0x0-0x3	0x0-0xFF	0x0-0x3	0x0-0xFF	0x0-0x3	0x0-0xFF	0x0-0x3	0x0-0xFF	0x6C

RX: No return message.

For all DoAs (MA=1 to MA=4), I_H should be set as the minimum current absorbed by the motor when driven at maximum speed (i.e. PWM set to 511) from the OPEN position. This current usually equals the motor current absorption right after the starting absorption peak (as shown in Fig. 14).

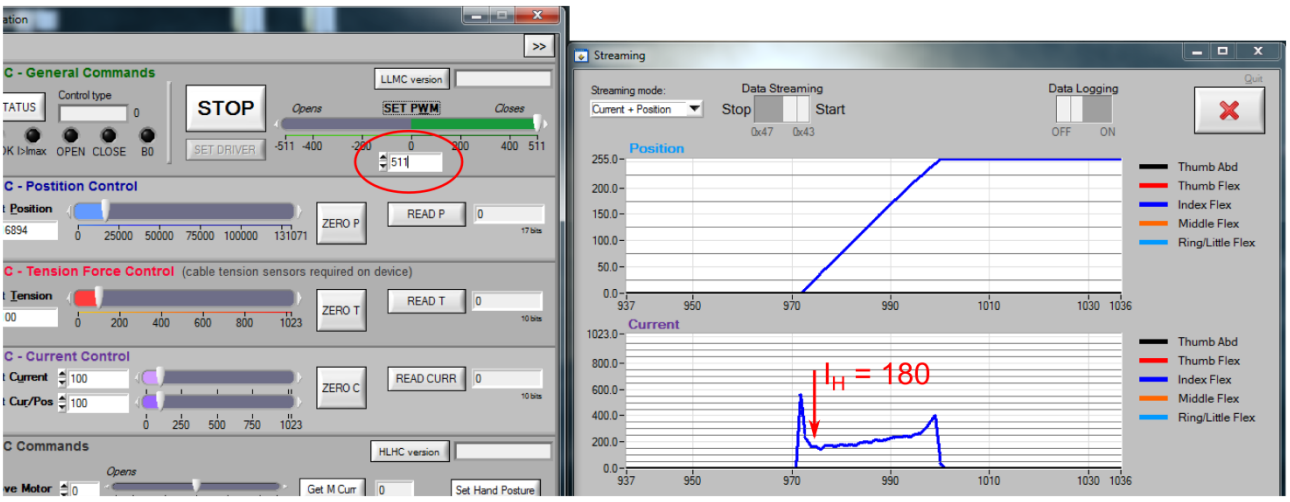

Fig. 14 Automatic grasp – Phases and timing during an automatic grasp.

5 LLMC commands

Each motor (or DoA) is directly actuated and controlled by means of a LLMC that achieves position control, tendon tension force control[7], current (torque) control, hybrid current/position control and monitors motor current absorption (ensuring a long-life operation of the motor). Each LLMC reads 1 tendon tension force[7], 1 motor current, 1 position (encoder) sensor outputs; resolutions are 10, 10 and 17 bits respectively. *The tension sensor is not provided for the thumb abduction/adduction DoF*[7].

The Low Level Motion Controllers of *IH2 Azzurra*, hence the single DoAs, are accessible through the set of commands described in this chapter. All these commands follow the packet format, described in Fig. 15. The packet size can be four or more bytes long, depending upon the nature of the command being sent to the hand. Each packet consists of an initial "start of packet" byte, followed by the LLMC Motor Address byte (to which the command is issued), followed by a "command" specifier byte, followed by zero or more "command data" bytes, and followed by the LLMC Motor Address byte.

⚠ The three PID commands are exceptional because only in these cases, the standard packet is followed by a termination byte (0xAA).

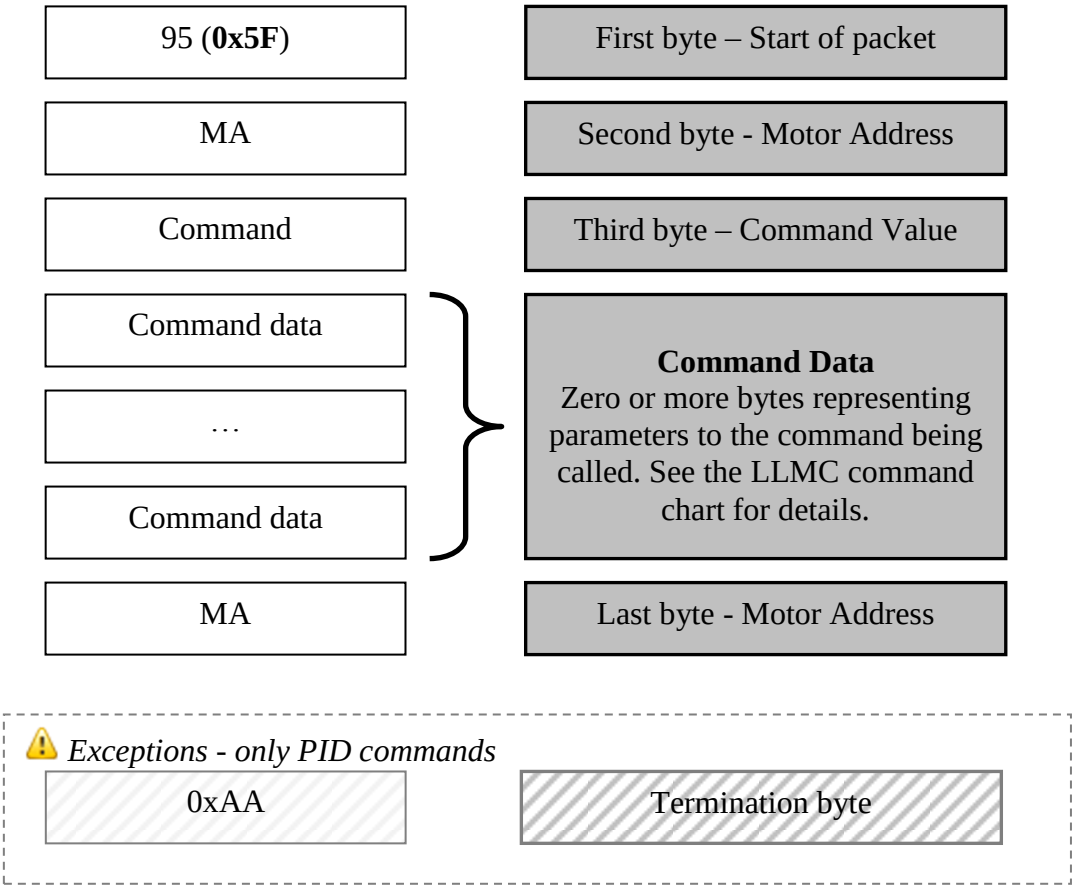

Fig. 15 – LLMC commands.

[7] Tendon tension force sensors are not available on all hands.

5.1.1 LLMC command description

Commands are divided based on which control mode they operate: General commands, Position control commands, Tension control commands[8], Current control commands.

Type	CMD Name	Description
GENERAL	STATUS	Reads STATUS BYTE of the LLMC (paragraph 2.4.3).
	STOP	Brakes motion of DoA
	MemPWMmax	Memorizes max PWM value (in E^2PROM)
	MemCURRmax	Memorizes max current allowed
	SETPWM/Speed	Sets motor PWM (speed feed-forward control)
	READ PWMmax	Reads max PWM value (from E^2PROM)
	READ CURRmax	Reads max current value (from E^2PROM)
	LowLevelVersion	Reads the version of the firmware of the LLMC
POSITION CONTROL	SETP	Sets desired position (17 bits format) – raw position
	READP	Reads position (17 bits format)
	ZEROP	Sets offset position for encoder
	PIDP ⚠	Memorizes PID parameters (in LLMC E^2PROM): Kp, Ki, Kd, ERRmax (max position error allowed)
	DUMPP	Reads PID parameters from E^2PROM
TENSION CONTROL	SETT	Sets desired tendon tension (10 bits format)
	READT	Reads tendon tension (10 bits format)
	ZEROT	Sets offset tendon tension
	PIDT ⚠	Memorizes PID parameters (in LLMC E^2PROM): Kp, Ki, Kd, ERRmax (max tension error allowed)
	DUMPT	Reads PID parameters from E^2PROM
CURRENT CONTROL	SETCURR*	Sets desired motor current (only for closure)
	READCURR	Reads motor current (10 bits format)
	ZEROCURR	Sets offset current
	PIDCURR ⚠	Memorizes PID parameters (in LLMC E^2PROM): Kp, Ki, Kd, ERRmax (max current error allowed)
	DUMPCURR	Reads PID parameters from E^2PROM
	SETCURRPOS*	Sets motor current (only for closure); if obstacle found (no more movement) the motor STOPS (with STATUS B4 set)

Table 3 LLMC Command list.

⚠ * The difference between SetCurr and SetCurrPos is that SetCurr imposes and maintains a precise current flowing into the motor (traditional current/torque control); SetCurrPos imposes a precise current (traditional current/torque control) but if an obstacle is found and the motor is no longer able to close, **the motor stops**.

⚠ The PIDP, PIDT and PIDCURR require the termination byte (cf. Fig. 15).

⚠ Kp, Ki, Kd and ERRmax are all encoded using 1 byte (0-255) for all control modes.

⚠ The commands included in this protocol are used for setting motion control PID parameters, setting current and speed limits to the motors, and other low-level setting commands. A non expert user might disregard this section and use the HLHC commands described in chapter 4.

[8] Tendon tension force sensors are not available on all hands.

5.1.2 LLMC commands chart

CMD Name	CMD Value	CMD Data length	Command Data bytes	Return data length	Return packet
STATUS	0x70	0	-	1	STATUS byte
STOP	0x71	0	-	0	-
MemPWMmax	0x72	2	PWMH, PWML	0	-
MemCURRmax	0x73	2	CURRH, CURRL	0	-
SETPWM/Speed	0x74	2	ByteH, ByteL	0	-
READ PWMmax	0x76	0	-	2	PWMH, PWML
READ CURRmax	0x77	0	-	2	CURRH, CURRL
LowLevelVersion	0x40	0	-	18	ASCII string (terminated by /NUL character)
SETP[9]	0x21+ ByteU*16	2	ByteH, ByteL	0	-
READP	0x22	0	-	3	ByteU, ByteH, ByteL
ZEROP	0x23	0	-	0	-
PIDP *	0x24	4	Kp, Ki, Kd, Error	0	-
DUMPP	0x25	0	-	4	Kp, Ki, Kd, Error
SETT	0x41	2	ByteH, ByteL	0	-
READT	0x42	0	-	3	ByteH, ByteL
ZEROT	0x43	0	-	0	-
PIDT*	0x44	4	Kp, Ki, Kd, Error	0	-
DUMPT	0x45	0	-	4	Kp, Ki, Kd, Error
SETCURR	0x61	2	ByteH, ByteL	0	-
READCURR	0x62	0	-	3	ByteH, ByteL
ZEROCURR	0x63	0	-	0	-
PIDCURR*	0x64	4	Kp, Ki, Kd, Error	0	-
DUMPCURR	0x65	0	-	4	Kp, Ki, Kd, Error
SETCURRPOS	0x66	2	ByteH, ByteL	0	-

* The PID commands require the termination byte (cf. Fig. 15).
An equivalent (eventually more readable) command chart table is presented in chapter 7.

5.1.3 Command data bytes format table

Data encoded using more than a byte should be sent/received as described in the following table.

Type	Resolution [#bits]	ByteU	ByteH	ByteL	Notes:
PWM	9	-	S x x x x x x B_8	B_7 B_6 ….. B_1 B_0	S=0 Open S=1 Close
Position	17	001 B_{16} 0001	B_{15} B_{14} ….. B_9 B_8	B_7 B_6 ….. B_1 B_0	in **SETP**
Position	17	x x x x x x x B_{16}	B_{15} B_{14} ….. B_9 B_8	B_7 B_6 ….. B_1 B_0	in **READP**
Tension	10		x x x x x x B_9 B_8	B_7 B_6 ….. B_1 B_0	
Current	10		x x x x x x B_9 B_8	B_7 B_6 ….. B_1 B_0	
CurrPos	10		x x x x x x B_9 B_8	B_7 B_6 ….. B_1 B_0	

[9] In the command SETP the 17th bit of the raw position is included in the command value (0x21). Details in paragraph 5.1.3.

6 Demo application

The demo application (available from Prensilia website) is able to issue all the commands *IH2 Azzurra* is capable of, which are presented in this user guide. The application is compatible with 32 bit or 64 bit machines with MS Windows installed as the operative system (Windows 2000, XP, Vista, 7, 8). To install the software simply double click on the setup.exe and follow the installation instructions. Once installed launch the application from Start → Programs → Prensilia Demo Application → Prensilia Demo Application.

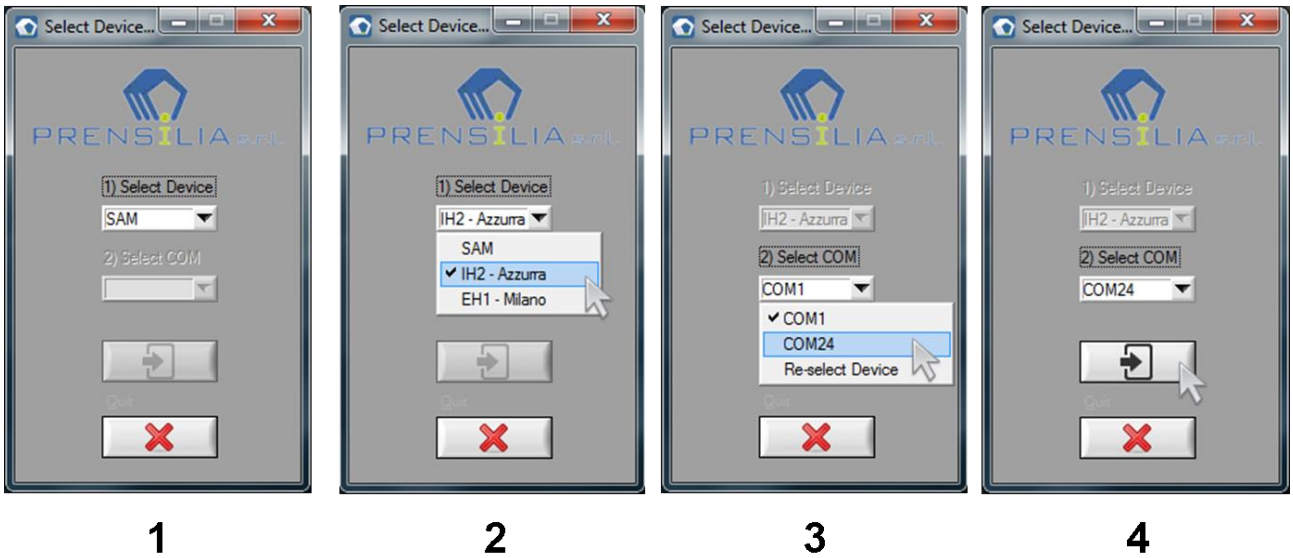

Fig. 16 Prensilia Demo Application - Select Device and COM panel.

Once launched the "Select Device and COM" panel will open (Fig. 16). Select your device (*IH2 Azzurra*) then select the COM port associated to the USB cable from the list of COM ports available on your PC. Once selected the button will be enabled and by clicking it the main panel will open (Fig. 17). The button quits the application.

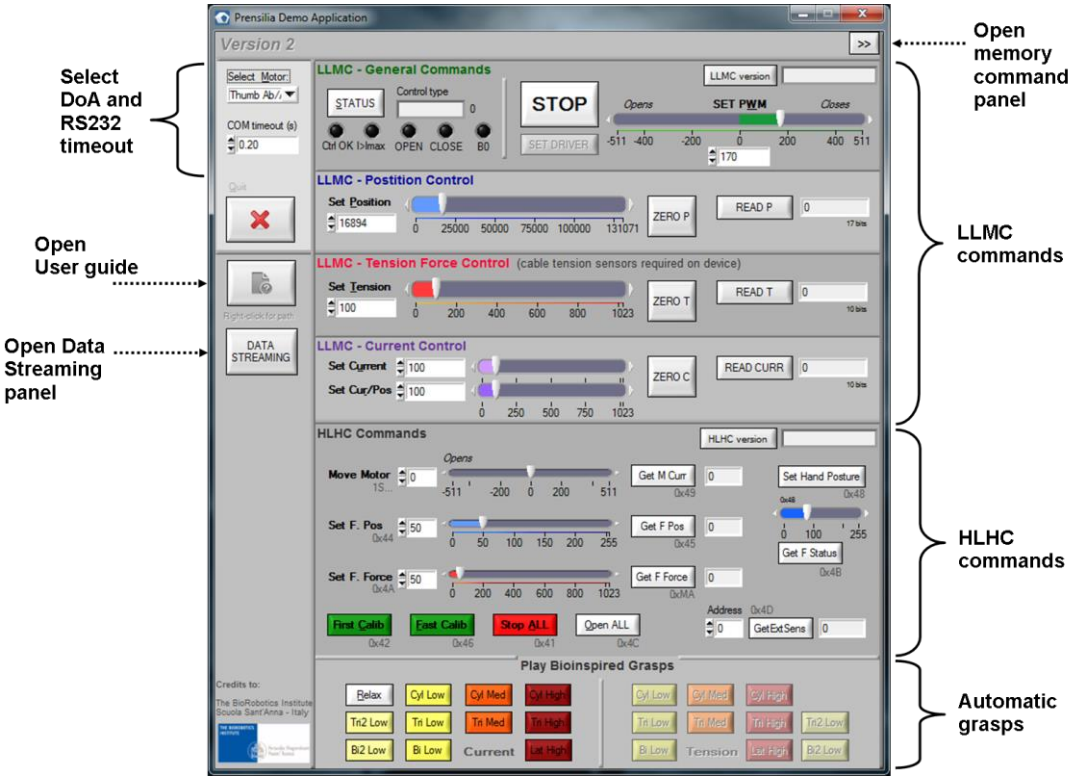

Fig. 17 Prensilia Demo Application – main panel.

The buttons on the main panel are self-explaining and have the same names of the commands described in this user guide. Next to each button that issues a HLHC command the hexadecimal code of the command is also written. Clicking on the button will open this user guide (if not renamed and saved in the correct path).

The button >> on the top right enlarges the panel and gives access to the memory commands, i.e. those commands that store values in the LLMC or HLHC non-volatile memories (
Fig. 18). A pop-up will warn the user about this event.

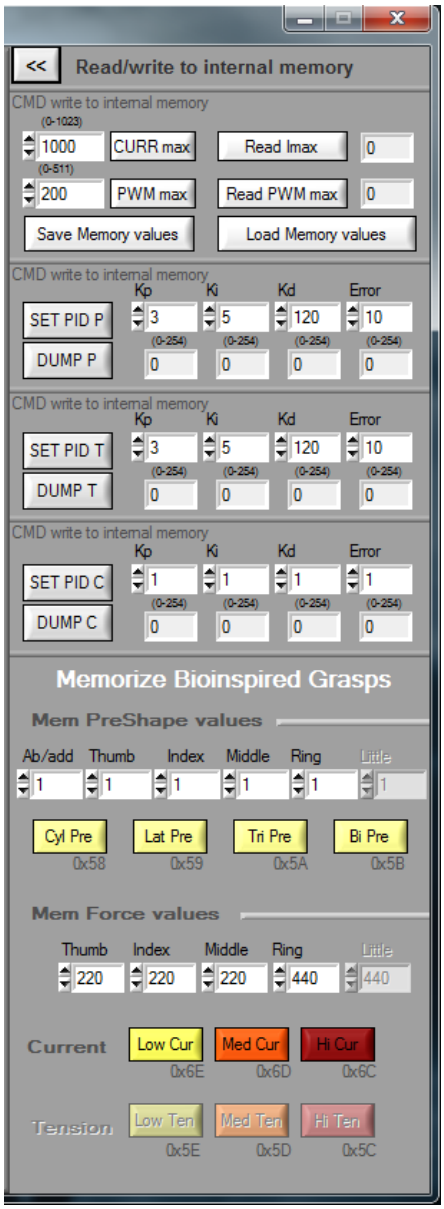

Fig. 18 Prensilia Demo Application – memory command buttons.

The function of the buttons is self-explanatory and consistent with the description reported in this application. In addition to the commands described above, the application is also provided with a "Save memory values" and a "Load memory values" buttons. They allow to quickly store/load the PID values for each motor and for each control type and the CURR max and PWM max thresholds into/from an external text file on the host controller PC. This comes in handy if *IH2 Azzurra* needs to be reprogrammed (e.g. in case of a firmware upgrade) speeding up the procedure of restoring the values inside the internal memory.

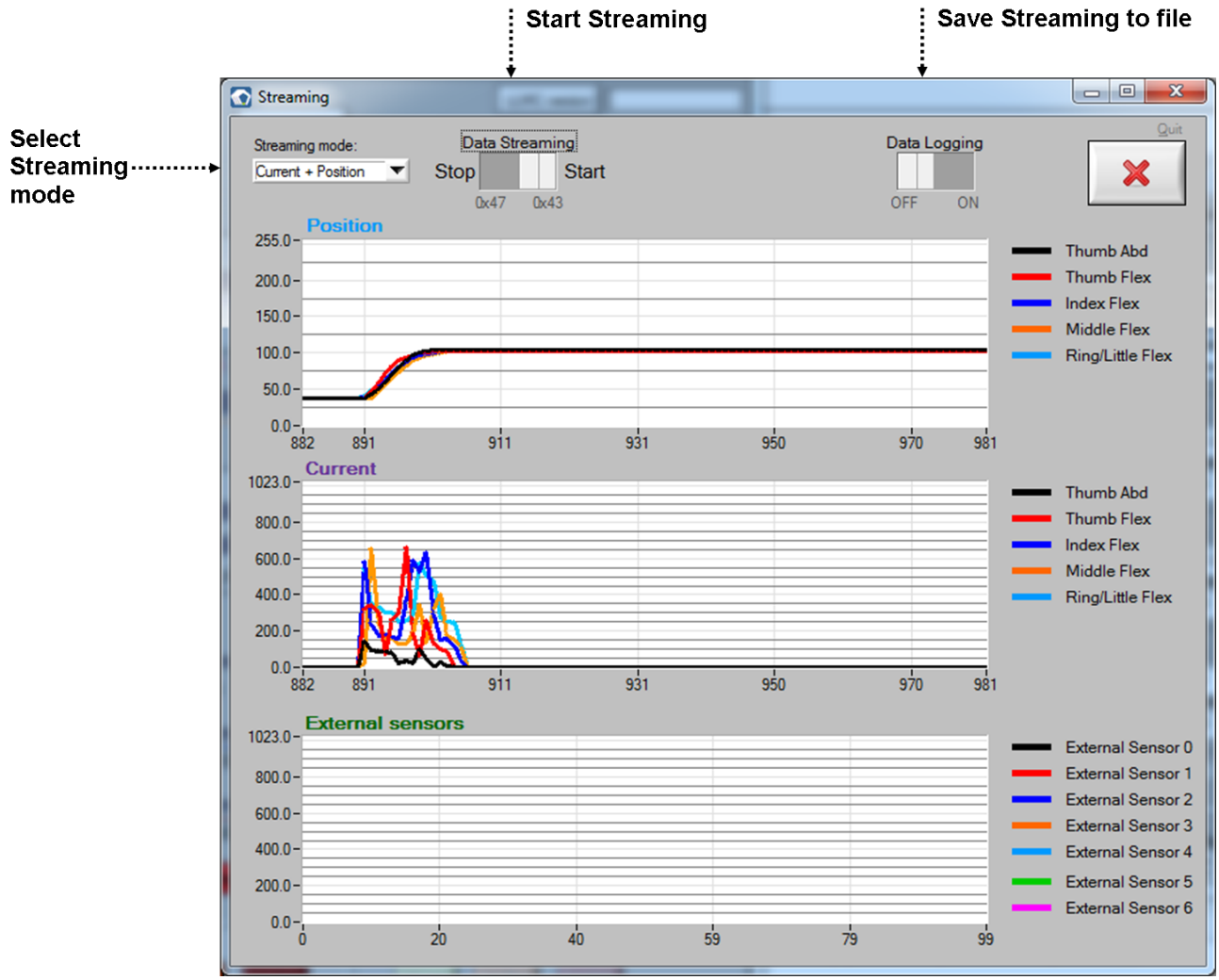

Fig. 19 Prensilia Demo Application –Streaming panel

The Streaming panel (Fig. 19) allows to check the Streaming Modes as explained in detail in section 4.3.5 on page 27.

Prensilia strongly suggests to install a RS232 sniffer on the host controller PC. A sniffer (or packet analyzer) is a computer program that can intercept and log traffic passing over a communication bus, in this case over the serial port to/from the hand. A sniffer will foster the development of the user application. Prensilia provides in the CD accompanying *IH2 Azzurra* a list of freely downloadable sniffing software. However several other applications can be easily found on the internet.

7 Resuming tables

	Command packet				Return bytes (if any)				
CMD Name	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
STATUS	0x5F	MA	0x70	MA	STATUS				
STOP	0x5F	MA	0x71	MA					
MemPWMmax	0x5F	MA	0x72	ByteH	ByteL	MA			
MemCURRmax	0x5F	MA	0x73	ByteH	ByteL	MA			
SETPWM/Speed	0x5F	MA	0x74	ByteH	ByteL	MA			
READ PWMmax	0x5F	MA	0x76	MA	PWMH	PWML			
READ CURRmax	0x5F	MA	0x77	MA	CURRH	CURRL			
LowLevelVersion	0x5F	MA	0x40	MA	18 bytes (ASCII string terminated by /NUL byte)				
SETP	0x5F	MA	0x21+ ByteU*16	ByteH	ByteL	MA			
READP	0x5F	MA	0x22	MA	ByteU	ByteH	ByteL		
ZEROP	0x5F	MA	0x23	MA					
PIDP	0x5F	MA	0x24	Kp	Ki	Kd	Error	MA	0xAA
DUMPP	0x5F	MA	0x25	MA	Kp	Ki	Kd	Error	
SETT	0x5F	MA	0x41	HSB	LSB	MA			
READT	0x5F	MA	0x42	MA	ByteH	ByteL			
ZEROT	0x5F	MA	0x43	MA					
PIDT	0x5F	MA	0x44	Kp	Ki	Kd	Error	MA	0xAA
DUMPT	0x5F	MA	0x45	MA	Kp	Ki	Kd	Error	
SETCURR	0x5F	MA	0x61	HSB	LSB	MA			
READCURR	0x5F	MA	0x62	MA	ByteH	ByteL			
ZEROCURR	0x5F	MA	0x63	MA					
PIDCURR	0x5F	MA	0x64	Kp	Ki	Kd	Error	MA	0xAA
DUMPCURR	0x5F	MA	0x65	MA	Kp	Ki	Kd	Error	
SETCURRPOS	0x5F	MA	0x66	HSB	LSB	MA			

STATUS Byte:

B7	B6	B5	B4	B3	B2	B1	B0
Control Type			Ctrl OK	FCA	FCB	I	MOVING
B7	B6	B5	Type	1: OK 0: not OK	1: open	1: close	1: I>Imax 0: I<Imax	1: moving 0: steady
0	0	0	STOP					
0	0	1	PWM Ext/Speed					
0	1	0	Position					
0	1	1	Tension (tendon)					
1	0	0	Current					
1	1	0	Current/Position					
1	1	1	COM ERROR					

MoveMotor	1 S MA3..MA0 X D8		D7..D0				S=1 closes, S=0 opens			
SetFingerPosition	0x44		Motor no.		Molt					
SetFingerForce	0x4A		T9 T8 x x MA3..MA0		T7..T0					
GetFingerPosition	0x45		Motor no.		Returns actual position					
GetFingerForce	0 0 xx MA3..MA0				Rx 10 bit force xxxxxx T9 T8 T7..T0					
GetMotorCurrent	0x49		Motor no.		Rx 10 bit current xxxxxx C9 C8 C7..C0					
GetExternalSensor	0x4D		Address of sensor		Rx 10 bit sensor xxxxxx S9 S8 S7..S0					
GetFingerStatus	0x4B		Motor no.		Rx: status byte					
FirstCalibration	0x42									
FastCalibration	0x46									
StopALL	0x41									
SetHandPosture	0x48	Pos0	Pos1	Pos2	Pos3	Pos4	0x48			
OpenALL	0x4C									
EnableStreaming	0x43	StreamMode								
DisableStreaming	0x47									
Automatic Grasps	0x6F	Type	Force	Duration						
Grasp Stepper	0x4E	Step	Type	Force	0x4E					
MemCylPreShape	0x58	Pos0	Pos1	Pos2	Pos3	Pos4	0x58			
MemLatPreShape	0x59	Pos0	Pos1	Pos2	Pos3	Pos4	0x59			
MemTriPreShape	0x5A	Pos0	Pos1	Pos2	Pos3	Pos4	0x5A			
MemBiPreShape	0x5B	Pos0	Pos1	Pos2	Pos3	Pos4	0x5B			
MemHighCurr	0x6C	CurrH1	CurrL1	CurrH2	CurrL2	CurrH3	CurrL3	CurrH4	CurrL4	0x6C
MemLowCurr	0x6E	CurrH1	CurrL1	CurrH2	CurrL2	CurrH3	CurrL3	CurrH4	CurrL4	0x6E

IH2 Firmware Version: HLHC (hlhc_26042016), LLMC (llmc_20052015)

V.le Piaggio, 32
56025 Pontedera (PI), Italy
info@prensilia.com
www.prensilia.com

IH2 - Self-Contained Robotic Hand *Basic* User Guide - Version 1.7
May 2016
Printed in Italy