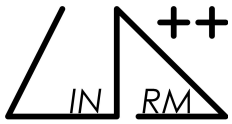


LINarm++

Affordable and Advanced LINear device for ARM rehabilitation

Deliverable D2.1

Control system architecture and components

Contractual delivery date	31.10.2015 (month 6)
Actual delivery date	31.10.2015 (month 6)
Version	1.0
Dissemination level	CO
Authors	Matteo Malosio (CNR) Alessio Prini (CNR) Matjaz Mihelj (UL) Janez Podobnik (UL) Andrea Crema (EPFL)

Table of contents

Executive summary

This document is intended to present the LINarm++ architecture and control system. It is designed to face and manage all the foreseen tasks of the LINarm++ platform as:

- control of the *LINarm mechatronic device* called *Linarm robot* in the sequel
- collect, manage and display all the system data
- define and represent rehabilitation tasks and targets to care givers and patients
- control functional electrical stimulation (FES) enabling the system to support hybrid rehabilitation therapies
- infer in run-time the level of assistance needed by the patient for a certain task in his particular and actual physiological condition
- render a virtual feedback to the patient consistent to activity performed and to the task given
- synchronize activities performed by the devices of the system

The system architecture presented here is intended to present devices, connections, communication protocols and how all the components interact to realize the foreseen functionalities activities. The term *device* stands for all the hardware and software (H&S) components needed to realize a defined set of activities. The term *connection* stands for all the H&S components (wires, protocols and so on) needed to realize the communication between two devices.

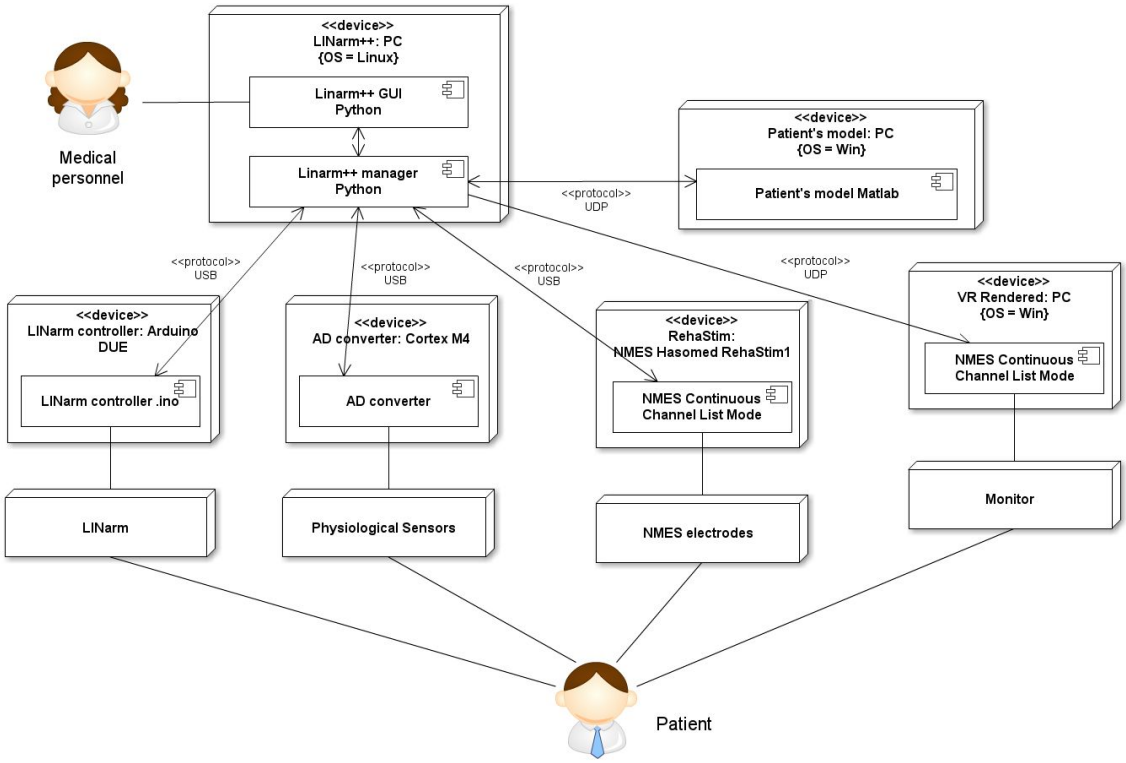
The system architecture and its functioning are presented in Chapter 1.

All the *devices* of the architecture are described in detail in Chapter 2.

Connections between each device are described in detail in Chapter 3.

1. System architecture

The underlying figure represents the LINarm++ system architecture



UML representation of the LINarm++ architecture.

The architecture is interfaced to the medical personnel through the LINarm++ device. It is a Python program running on a PC made up of two main layers. Being Python-based it is intrinsically multi-platform.

The upper layer, the *LINarm++ GUI*, is a GUI (Graphic User Interface) which allows the user to control all the system functionalities and graphically represents all the system data .
The lower layer, the *LINarm++ Manager*, has the aim of managing the LINarm++ system coordinating all the system's devices according the selected control modalities and selected functionalities.
It receives streams of different data, as kinematic data, physiological data, level of assistance data, dispatches them to other devices coherently with their use and applies control logics and functioning modalities..

The *LINarm controller* is an Arduino DUE board running a C++ program. This board is a low-cost general purpose featuring a cortex M3 microcontroller. The C++ software running on this board controls in real-time the *LINarm* device according to a set of commands exchanged with the *LINarm++ manager*.
The *LINarm* node is a variable-stiffness-actuated robotic device performing linear movements. It is widely illustrated in D3.1 deliverable.

A set of physiological data are measured by a set of physiological sensors and sent to the *LINarm++ manager* by a signal acquisition board. Measured quantities are heart rate, skin conductance and skin temperature. An additional measure refers to the grasping force. These data are acquired by a signal acquisition board which sends collected data through a USB protocol connection.

All data are collected by the *LINarm++ Manager* and made available to the *patient model* device. It is a Matlab program aiming at estimating in run-time the patient state and the recommended assistance level. As Python scripts also Matlab scripts are portable allowing realize a platform-independent software architecture. All data exchanged between the *LINarm++ manager* and the *Patient model* are sent through an UDP connection.

The assistance level evaluated by the *Patient model* is exploited by the *LINarm++ Manager* to determine both the robotic assistance and the FES assistance to be given to the patient. The robotic assistance is given by the *LINarm* device. The *RehaStim* node is responsible of giving FES assistance and consists of an electrostimulator in charge of generating stimulation currents according to commands sent over a USB connection by the *LINarm++ Manager*.

The *VR Renderer* node and the *Monitor* are helpful generate the graphical representation of the task to be carried out by the patients in the form of a game, involving the patient in performing a defined task.

All the devices are integrated to a render a multisensorial experience while performing and administering rehabilitation tasks and protocols. More detailed devices descriptions are given hereafter.

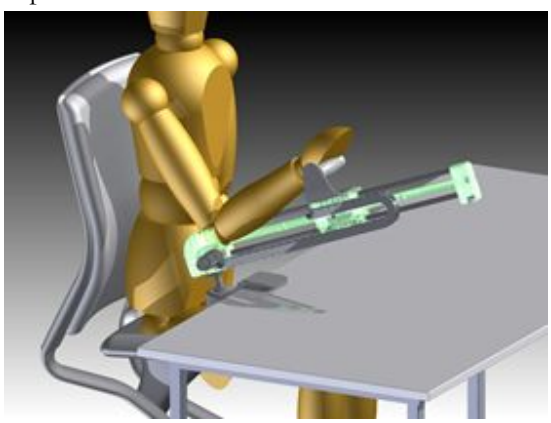
2. Devices

In this section all the devices and their functionalities of the LINarm++ architecture are described in detail. For each device, the corresponding node is highlighted in the system architecture diagram.

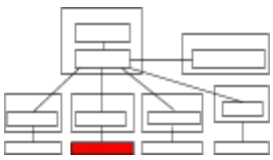
LINarm

LINarm is the mechatronic device of the *LINarm*++ experiment. During the experiment a redesigned version is being developed, namely *LINarm2*, and its details are illustrated in *D3.1 - Mechatronic device design*.

It is grounded through an orientable spherical joint and features an instrumented handle held by the patient. The device is equipped with two actuators, position sensors and switches to control its functioning and movements. All these electromechanical equipments are connected to an Arduino board in charge of its real-time control. Additionally, it will be equipped with a communication channel acquiring data from a set of sensors installed on the handle. These signals are transmitted through a proper cable chain installed in parallel to the linear axis..



LINarm2 mechatronic device installed on a desk.

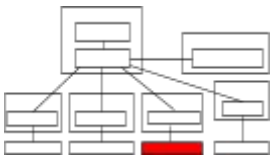


Physiological sensors

A set of physiological sensors are installed in the system to estimate the state of the patient and, subsequently tune the level of assistance. Physiological sensors included in the device are:

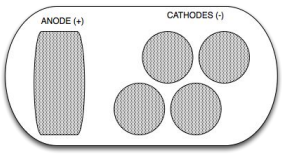
- Heart rate sensors
- Skin conductance sensors
- Skin temperature sensors
- Grasp force sensors

For more detailed description please refer to *Deliverable D4.1*.



NMES electrodes

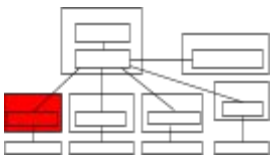
The electrode arrays used for targeting the chosen muscular districts are obtained by embedding four standard electrodes into a fabric support, and by providing a convenient cabling system, able to provide low encumbrance, and ease of wearability. Since the electrodes are arranged as a small matrix, suboptimal alignment can be improved by software choice of the best combination electrodes. For a more detailed description please refer to D 5.1.



Scheme of a matrix of electrodes.

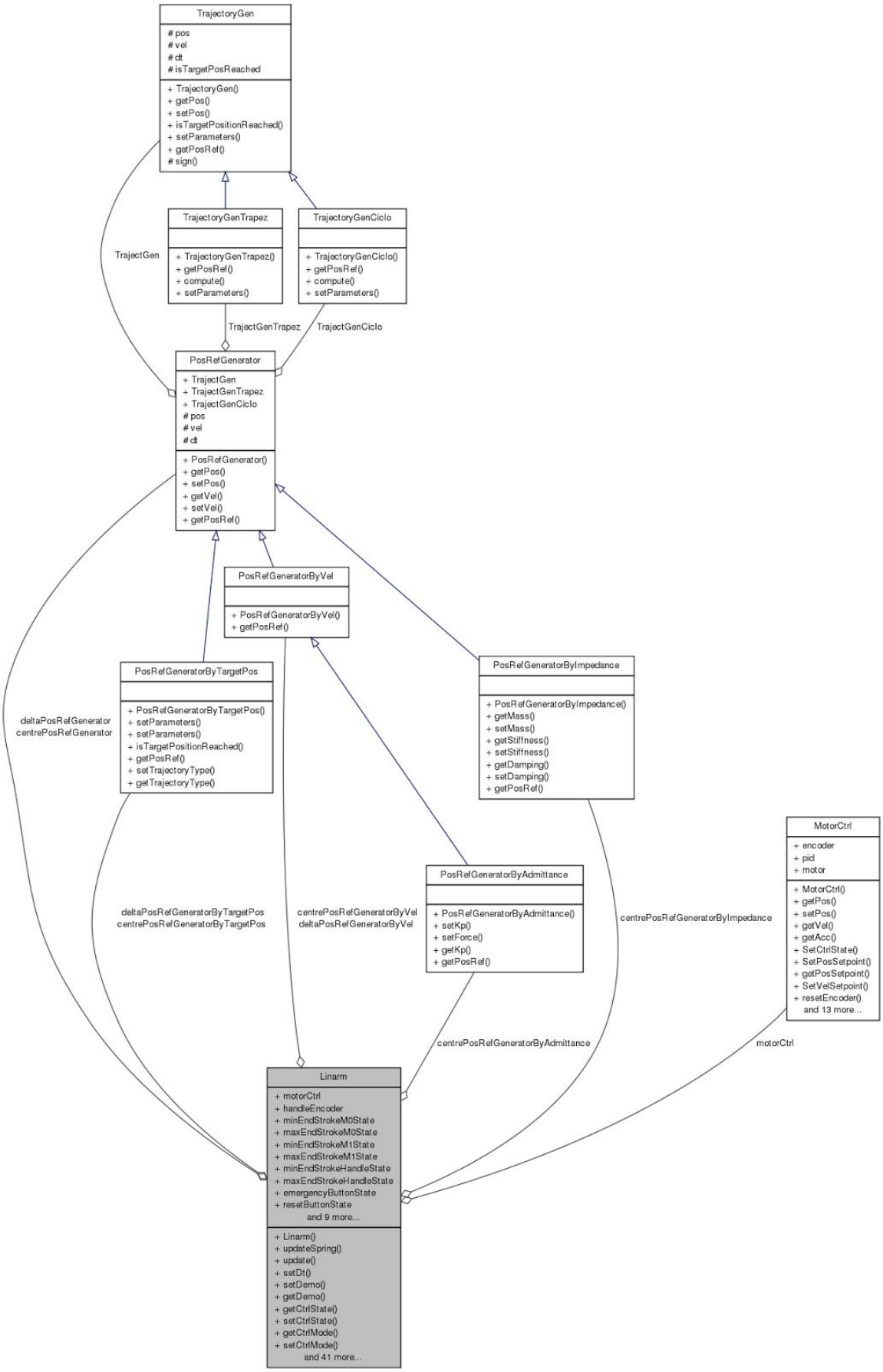
Monitor

LCD monitor with standard resolution for displaying graphical representation of the task virtual environment.



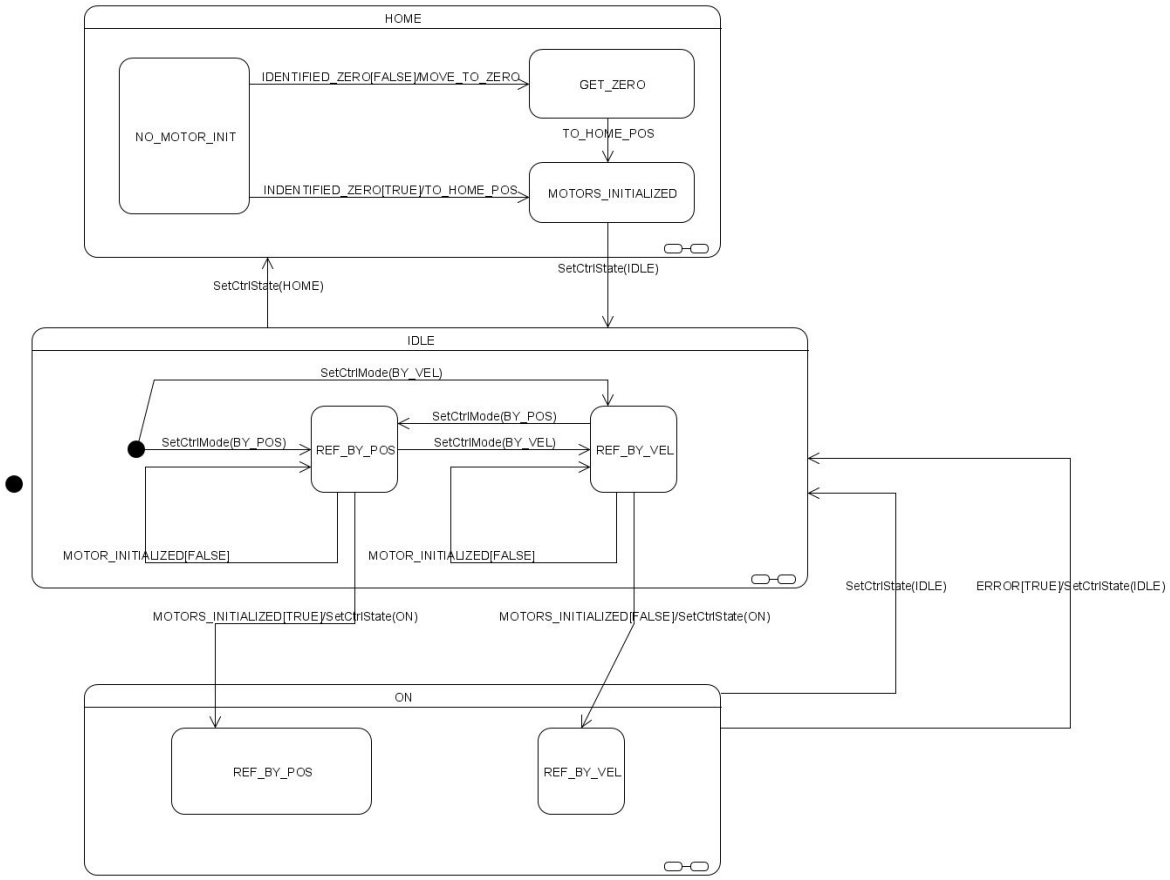
LINarm controller

The *LINarm conroller* is an Arduino board Due running a composed a real-time C++ software. Arduino Due is a general purpose board based on Atmel SAM3X8E ARM Cortex-M3 CPU, with a series of analog and digital I/O port, exploited to control the LINarm actuators and sensors.(https://www.arduino.cc/en/Main/ArduinoBoardDue)The software architecture of this device is composed by a series of object that give a software abstraction of the *LINarm* physical element like motor, encoder, VSA spring ecc. The figure hereafter represents a scheme of the software architecture.



UML software architecture representation.

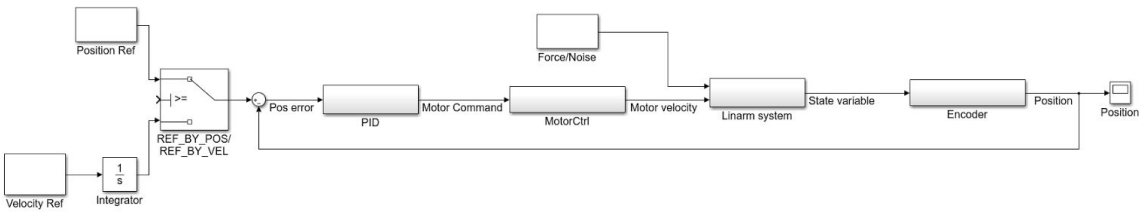
As depicted, the main object is *Linarm*. It includes a finite state machine necessary to control the *LINarm* robot in all the working phases. The finite state machine is depicted in the follow figure.



Finite State Machine of the LINarm device.

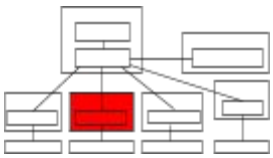
The Finite State Machine is characterized by three main states.
The IDLE state is the initial state of the device. In this state it's possible to select the movement control mode(BY_POS or BY_VEL) of LINarm. In this state motors are off.
In the HOME state LINarm perform an homing procedure, automatically manage by LINarm controller. During this phase, encoders are reset exploiting end-strokes switches and a predetermined home position is reached at the end of the home procedure. In order to carry out the home procedure motors are switched on. In this phase MOVE_DELTA_TO, MOVE_CENTRE_TO or SET_DELTA_VEL and SET_CENTRE_VEL motion commands are used to control the LINarm movement. The names DELTA and CENTRE refer to the VSA spring. DELTA refers to the distance between two end point of the spring, CENTRE refers to the equilibrium point of the VSA. These functions are transformed in commands for the LINarm motors.

The real-time motion control is executed according to the control loop hereafter represented.



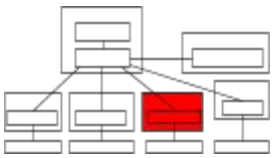
Real-time motion control scheme.

The main control consists of a position control loop. If the JOG motion control modality is selected (REF_BY_VEL mode selected) the reference velocity is integrated by time to obtain a position target to be used in the position control loop.
Sensors data, states and modes of the device are sent to *LINarm++ Manager*.



Physiological sensor acquisition board

Analog values from four sensors (force, heart rate pulse, skin conductance and temperature) are sampled by the STM32F4 12-bit ADCs at 100 Hz. Value for each sensor is represented by 2 bytes that are sent over serial communication to PC where signal processing is performed. Signal processing will be run on same computer as patient model.

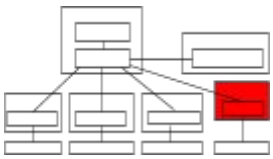


Rehastim

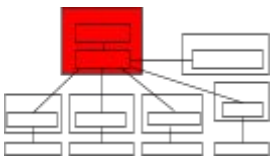
The Rehastim One (Hasomed GmbH, Magdeburg, DE) is a programmable eight channels electrical stimulator for clinical and research purposes. The ScienceMode, provided by Hasomed as a firmware extension of the standard device, is a protocol for the interface between the RehaStim and an external PC, which allows external control of the RehaStim in order to generate stimulation pulses.

Hasomed RehaStim electrostimulator.

The stimulator is controlled using the Continuous Channel List Mode, and the creation of complex patterns is greatly simplified. Additional hardware is included in the overall system to allow a simple interfacing of the stimulator with the wearables, and to minimize the overall numbe of needed cables. more detailed information is included in the deliveravle D5.1.



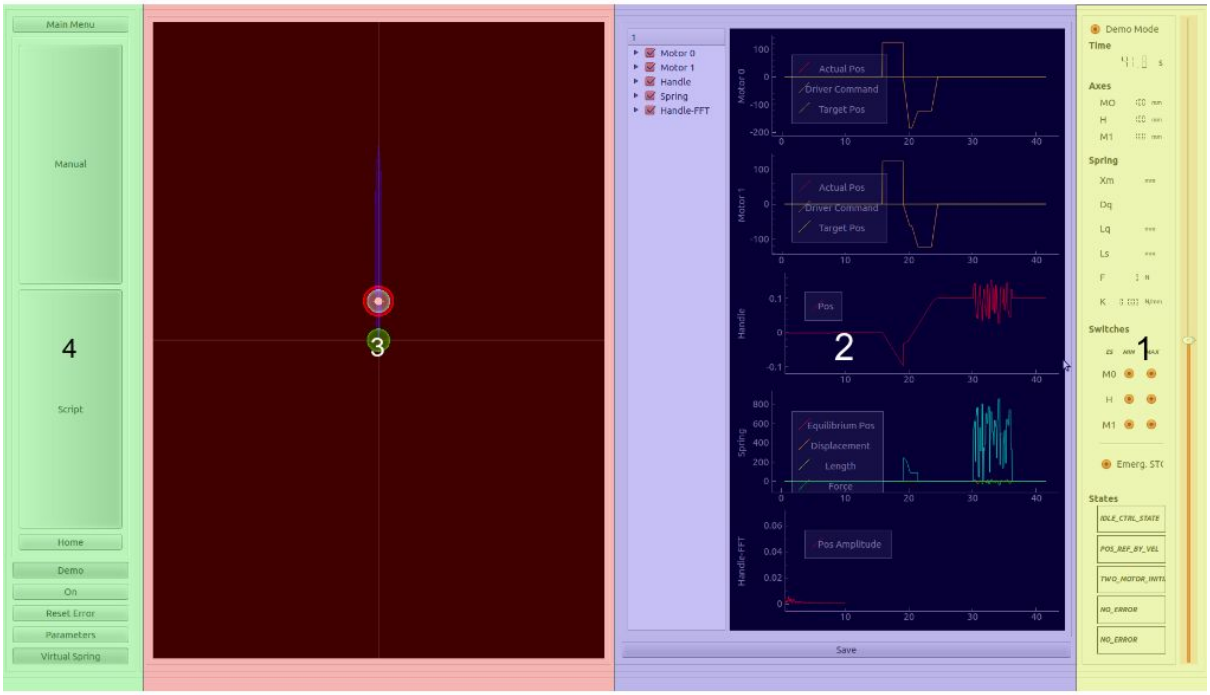
Graphical representation of the task virtual environment will be designed in Unity3D. Unity3D is cross-platform game engine used for developing 3D or 2D video games and graphical virtual environments. Graphical virtual environment can be compiled into stand-alone executable for Windows and Linux operating system.



.

As previously introduced, the LINarm++ device consist of two layers; *LINarm++ GUI* layer, interface to the medical personnel, and *LINarm++ Manager* layer, in charge of collecting data and controlling the functionalities of the overall architecture.

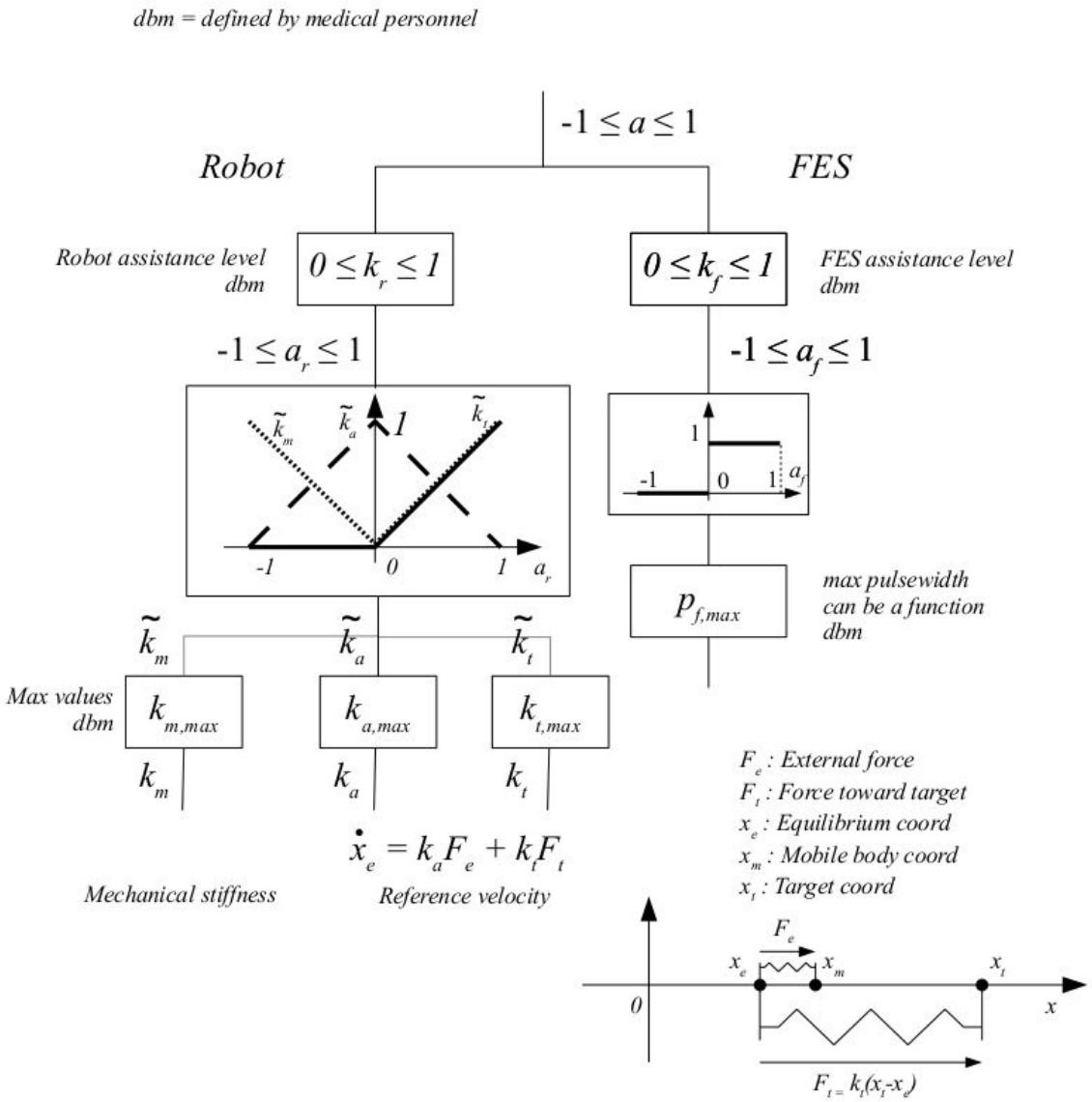
A preliminar screenshot of the GUI is reported hereafter.



LINarm++ Graphical User Interface.

It is written in Python language and exploits the PyQt4 library to realize graphical widgets and forms. The graphical interface is divided in four main frames fulfilling distinct functionalities.
Frame 1 groups numerical values of instantaneous system data. In its lower side control states and eventually arose errors are reported.
Frame 2 includes a set of 2D graphs. Available data can be selected and shown in run-time through the PyQtGraph library. In the lower side a Save button allows to save data.
Frame 3 includes a 3D simplified representation of the mechatronic system. This section is only a representation useful to medical personnel and is useful to test control algorithms and software functionalities during the development phase, simulating the device functioning, and to represent targets of the exercises.; the *VR-renderer* device is synchronized to represent exercises in a more pleasant way for the patient.
Finally, frame 4 includes commands and functionalities to control the LINarm++ system. On the upper side there are the main commands to control the LINarm++ system's devices. On the lower side there are buttons to select states and modes. A peculiarity of this section is represented by the *script* mode. In this mode it is possible to edit and execute a task consisting of a set of commands sent to LINarm++ devices. In this mode it's possible to select and customize the task to be executed .

Selections done on the *GUI layer* are trasformed in actual commands by the *LINarm Manager*. Even this layer is written in Python language. This peculiarity made the system portable also on platform different from Linux OS. *LINarm++ Manager includes* all the functionalities required to communicate with system devices. It can receive/send data from/to each device through proper communication protocols (refer to Chapter 3)..
By the LINarm++ Manager it's even possible define a list of parameter useful to split the assistance between the FES device and LINarm robot device. The following scheme help to understand how LINarm++ Manager does this.

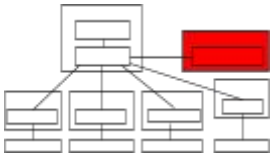


LINarm++ Manager receives the assistance level *a* calculated by the patient model (described in next section) according to physiological and kinematic data. This level is processed to get two quantities (*ar* assistance robot, i.e. LINarm, level and *af* assistance FES level) by *kr* anf *kf* parameters. According to specific patient's needs, the medical personnel define $0 \le k_r \le 1$ (robot assistance gain) and $0 \le k_f \le 1$ (robot assistance gain) patient by patient through the LINarm GUI.

Robot assistance is managed by coefficients *km*, *ka*, *kt*.

km refers to mechanical stiffness of LINarm VSA. *ka refers* to LINarm admittance control . *kt* refers to an additional active impedance of LINarm that take in account the difference between the handle position and target position for the task given to the patient.

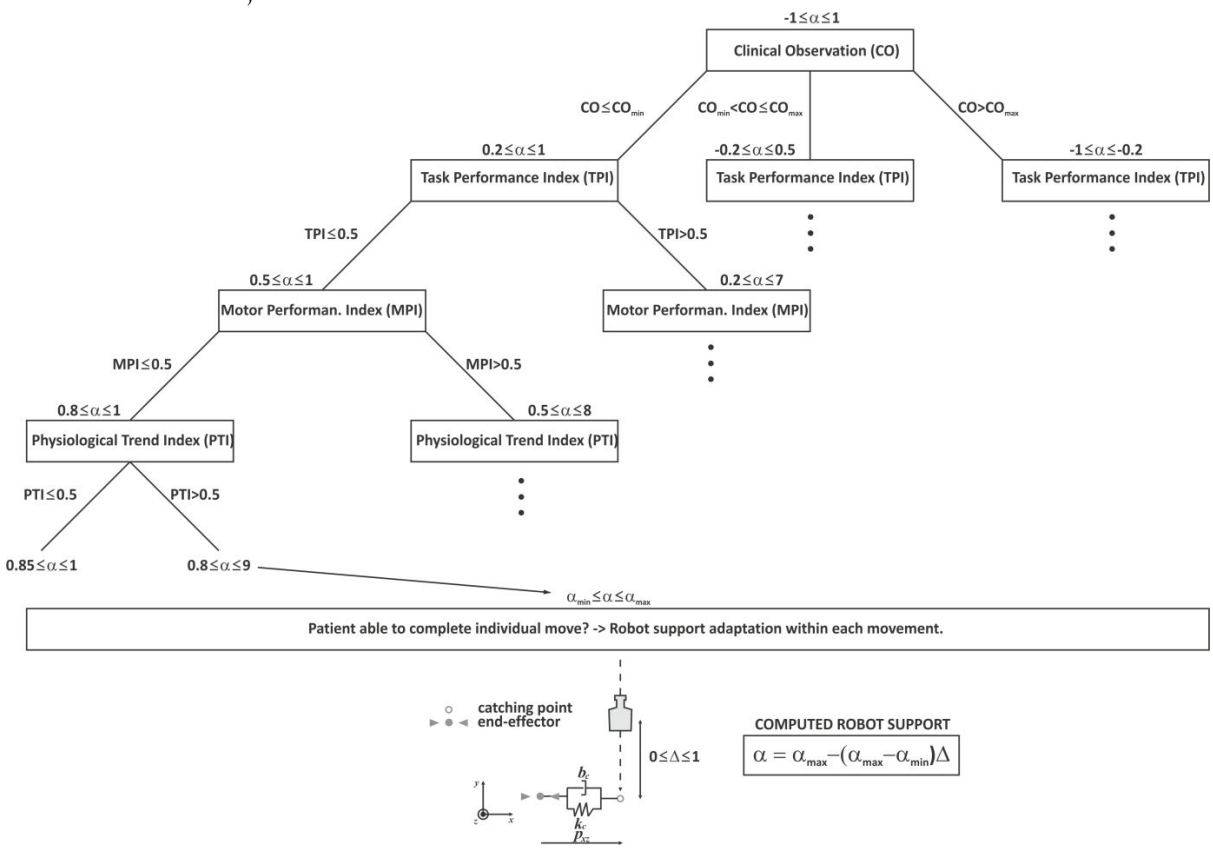
km, *ka*, *kt* are defined by the medical personnel through the GUI layer.



Patient model

Decision tree is an algorithm which uses tree-like model of consecutive decisions to determine the outcome. It is constructed of internal nodes and branches which branch from upper level nodes to lower level nodes. Each node contains a branching condition for one of the features and intermediate outcome. Last node is called leaf, which contains final outcome. Decision tree can be pruned by removing leafs (lowest level of nodes) and next lowest level of nodes become leafs and their corresponding intermediate outcomes become final outcomes.

Decision trees have advantage over similar decision algorithms over others algorithms because they represent transparent and understandable hierarchy of decision rules, and are therefore simple to understand, interpret and adapt. The other big advantage is that decision trees can be constructed using minimum learning set of data by a human expert who determines the structure and decision rules, which can be later adjusted when new data is available.



Decision tree for determining the level of robot support.

Patient model will be constructed as a decision tree with five layers (see previous Figure). The inputs to the decision tree are the following parameters:

- patient's clinical assessment scores,
- patient's task performance (scores, ability to complete single subtask, time to complete single subtask),
- patient's motor performance (force, velocity, power, smoothness, …), and
- patient's physiological assessment (heart rate, skin conductance, peripheral skin temperature).

The output of the decision tree is the value α that defines the robot support. At the input to the top layer the robot support α can have any value from full assistance ($\alpha = 1$) to full resistance ($\alpha = -1$), thus $-1 \leq \alpha \leq 1$. Each layer limits α values to a subset of $-1 \leq \alpha \leq 1$. In the final layer the parameter α gets a single scalar value that is sent to the robot controller.

The top layer has only one node with three branches. This node gives the highest priority to decisions of physicians and therapists that score patient's abilities with clinical scores. Based on his/her clinical scores the patient is assigned to one of the three groups: 1) the patient requires robot assistance, 2) the patient can work against robot resistance and 3) the patient is somewhere between robot assistance and resistance. At the top layer branching depends on the selected clinical score used for patient evaluation.

The second layer has three nodes with two branches each. The decisions on this layer are based on the Task Performance Index (TPI). The TPI is a variable normalized between 0 (poor performance) and 1 (excellent performance) and is computed from the results of the obtained in the training task. The exact model to compute the TPI will be determined later. Branching thresholds will be determined based on the empirical data obtained during training sessions with patients. The setting of thresholds can also be left to therapists.

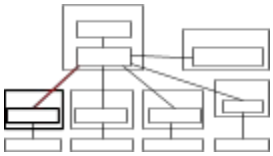
The third layer has six nodes (each node in the second layer has two branches) with two branches each. The decisions on this layer are based on the Motor Performance Index (MPI). The MPI is a variable normalized between 0 (poor performance) and 1 (excellent performance) and is computed from the measurements of parameters related to physical human-robot interaction. The exact model to compute the MPI will be determined later. Branching thresholds will be determined based on the empirical data obtained during training sessions with patients. The setting of thresholds can also be left to therapists.

The fourth layer has twelve nodes (each node in the third layer has two branches) with two branches each. The decisions on this layer are based on the Physiological Trend Index (PTI). The PTI is a variable normalized between 0 (no trend or negative trend of physiological signals) and 1 (positive change of physiological signals) and is computed from the physiological measurements. The exact model to compute the PTI will be determined later. Branching thresholds will be determined based on the empirical data obtained during training sessions with patients. The setting of thresholds can also be left to therapists. Since physiological measurements are the least reliable compared to other measurements the influence of PTI on the robot support is the smallest.

The fifth layer transforms the range of possible robot supports determined in the previous four layers into a single value to be sent to the robot. The single value is computed based on the performance related to a single subtask (e.g. movement to catch an object). If the patient is able to complete the subtask with the minimal determined level of support, he/she is allowed to do so. Otherwise, the support gradually increases toward the maximal determined level of support. In this case the minimal and the maximal level of support are only a subset of $-1 \leq \alpha \leq 1$, which was determined in the first four layers.

3. Connections

In this section the implementation details of the communication protocols are reported.

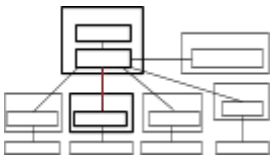


LINarm++ manager <-> LINarm controller

All communications are managed by the linarmApp object previously presented in LINarm controller section. The communications with the LINarm++ Manager are realized by the *CmdMessenger* library (http://playground.arduino.cc/Code/CmdMessenger), originally implemented to realize callback-based communications between Arduino and C# software. CNR ported the C# implementation to the Python language. Messages are formatted as follows:

```
Cmd Id, param 1, param 2, … , param N;
```

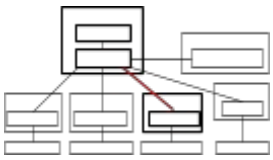
CmdMessenger supports callback function that realize one relationship between ID and related function. There are two main data types: *incoming data* and *sent data*. *Incoming data* are sent by *Linarm++ Manager* to *LINarm controller* and *sent data* are sent from LINarm controller to the LINarm++ Manager. Due to the multithread structure, data consistency is guaranteed by properly implemented mutexes and lockers.



LINarm++ manager <-> Physiological sensor acquisition board

Communication protocol between LINarm++ manager and Physiological sensor acquisition board will be standard serial UART communication protocol. Each package will contain 20 bytes and will be send with frequency of 100 Hz, requiring a 16kbit communication bandwidth.

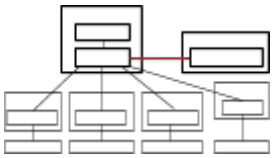
Header	Heart rate	Skin conductance	Skin temperature	Force sensor	Additional data
2 bytes	4 bytes	2 bytes	2 bytes	2 bytes	8 bytes



LINarm++ manager <-> Rehastim

The communication between the LINarm++ Manager and the RehaStim module in charge of controlling the FES electrodes is done, as previously introduced, exploiting the Continuous Channels List mode through a USB connection. This control mode delegates to the RehaStim the hard real-time control of stimulation patterns, leaving the LINarm++ Manager the possibility to change its parameters asynchronously and without real-time requirements.

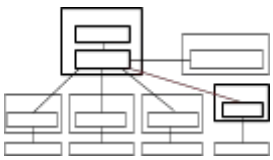
To this purpose, a communication driver (`RehaStim1.py`) has been implemented in Python and allows to control stimulation parameters by a generic Python program. It will allow to develop stand alone programs and will be embedded in the LINarm++ manager to synchronously control both the LINarm robot and the FES system by a single node. The driver has been implemented according to the RehaStim ScienceMode specifications. In order to generate customized profiled stimulation patterns a proper higher-level class has been developed to interpolate a set of values as function of desired parameters, e.g. time or trajectory normalized coordinate.



LINarm++ manager <-> Patient model

Communication protocol between LINarm++ manager and Patient model will be UDP communication protocol. Each package will contain 40 bytes and will be sent with frequency of 100 Hz, requiring a 32kbit communication bandwidth.

Header	Heart rate	Skin cond.	Skin temp	Force sensor	Interaction force	Robot vel.	Robot position	Target position	Task data	Additional data
2 bytes	4 bytes	2 bytes	2 bytes	2 bytes	2 bytes	4 bytes	2 bytes	4 bytes	8 bytes	8 bytes



Patient's model <-> VR Renderer

Communication protocol between LINarm++ manager and VR Renderer will be UDP communication protocol. Each package will contain 32 bytes and will be sent with frequency of 50 Hz, requiring a 12.8kbit communication bandwidth.

Header	Force sensor	Interaction force	Robot velocity	Robot position	Target position	Task data	Additional data
2 bytes	2 bytes	2 bytes	4 bytes	2 bytes	4 bytes	8 bytes	8 bytes