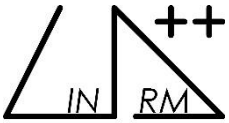


LINarm++
Affordable and Advanced LINear device for ARM rehabilitation

Deliverable D2.2
Control system

Contractual delivery date	31.07.2016 (month 15)
Actual delivery date	01.08.2016 (month 15+1d)
Version	1.0
Dissemination level	PU
Authors	Alessio Prini (CNR) Matteo Malosio (CNR)

Table of Contents

Executive summary

This report deals with the structure and the implementation of the control managing the whole LINarm++ platform. The report focuses on details about the actual implementation of the control, according to the design guidelines drawn in D2.1. Section 1 summarizes the overall control structure architecture, highlighting the tasks in charge to the different nodes of the system. Section 2 focuses on the implementation of the control architecture, describing how the ROS system enabled a straightforward and modular integration of the nodes. Section 3 describes in details the nodes of the system and how they have been implemented in details. Section 4 reports an example of use of the script-based programming system, enabling the integration of the signals of the system to realize rehabilitation tasks. The Appendix reports the manual of the overall control architecture.

1 Control architecture

The overall control architecture of LINarm++ (Fig. 1) is designed to face and manage all the foreseen rehabilitation tasks within the project. In particular:

- to control the LINarm2 mechatronic device;
- to collect, manage and display all the system data;
- to define and represent rehabilitation tasks and targets to care givers and patients;
- to control FES enabling the system to support hybrid rehabilitation therapies;
- to infer in run-time the level of assistance needed by the patient for a certain task in his particular and actual physiological condition;
- to render a virtual feedback to the patient consistent with the performed activity and the given task;
- to synchronize activities performed by the devices of the system.

The medical personnel may control and supervise all the system through the LINarm++ GUI (Graphic User Interface). The LINarm++ manager is in charge of coordinating all the sub-modules according the selected control modes and functions. It receives streams of different data, as kinematics, physiological parameters and level of assistance, dispatches them to other devices coherently with their use and applies control logics and functioning modes.

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.

The control system as a whole is able to constantly update rehabilitation parameters with the aim of fulfilling the needs of the patient during the therapy, estimating how the user is behaving during a single training session and evaluating changes during the entire training period.

All data collected by the LINarm++ manager are made available to the Patient's model device in charge of estimating the patient's state and the required assistance level. The assistance level is exploited to determine both the robotic assistance and the FES assistance to be given to the patient.

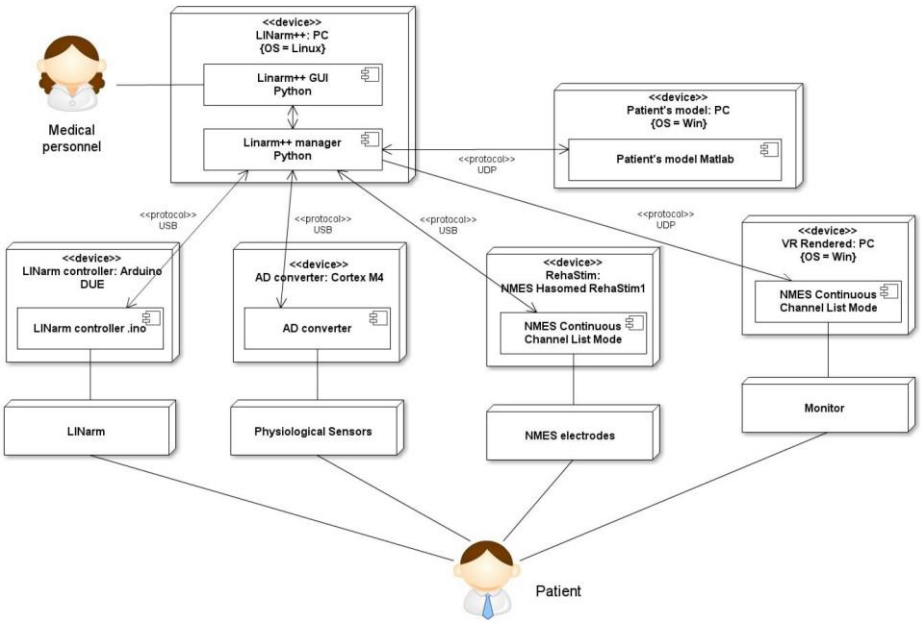


Fig. 1: UML representation of the LINarm++ architecture.

The LINarm++ Manager is in charge of tuning the assistance provided by both the LINarm2 device and the NMES system according to the assistance level a calculated, as previously described, by the patient model. As schematized in Fig. 13, the assistance level a is used to calculate the robot assistance level a r and the FES assistance level a f , exploiting the two constant k r and k f , respectively. The medical personnel define $0 \leq k_r \leq 1$ (robot assistance gain) and $0 \leq k_f \leq 1$ (FES assistance gain), according to specific patient's needs and impairment. The robot assistance a r is used to determine three parameters which characterize the assistance of the mechatronic device: k m is proportional to the mechanical stiffness of the VSA embedded in LINarm2, k a is the gain of the admittance-based control, k t is the gain of the assistive controller, in charge of assisting the patient to reach the target represented on the virtual scenario. The FES assistance a f is used to determine the pulsewidth defining the stimulation intensity.
In conclusion, all the parameters defining both the mechatronic and the FES assistance are derived by the assistance level, inferred by the patient model, in turn defined by a set of continuously updated kinematic and physiological parameters, and game scores.

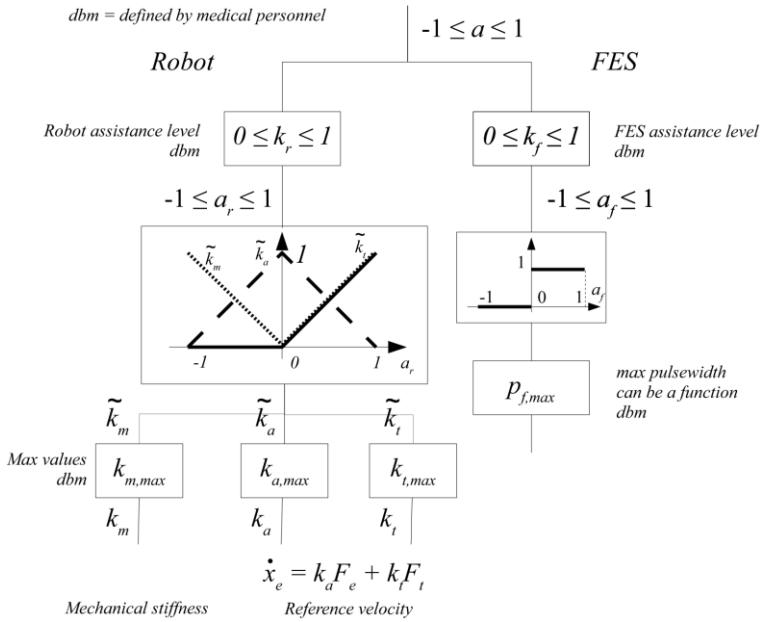


Fig. 2: Scheme of the assistance level tuning algorithm.

2 Control Implementation

From the description given above it's clear that the Linarm++ Manager software must be able to communicate and manage a significant number of modules. For this reason a sort of distributed system, with different processes, each able to manage a Linarm++ module, is the better choice. In this distributed system, every single process must be able to communicate with any other node through well defined types of message.

For this reason the ROS framework has been used to develop the Linarm++ Manager software architecture. ROS (https://en.wikipedia.org/wiki/Robot_Operating_System, http://wiki.ros.org/) is a collection of software framework, created to help the robotic software development and it is more and more becoming a standard in robotics.

ROS framework is able to provide the standard operating system services. For example in ROS every process is represented by nodes. ROS is able to manage the execution of different nodes and to manage the communication between these nodes. The communication among nodes can be performed by message defined by the developer through two main types of transport system.

The first type is the *topic*. A message can be published to a topic by a node. All the other nodes interested in a certain kind of data will subscribe to the appropriate topic. In this manner the production and consumption for the data are totally decoupled.

The other type of transport system is represented by the service. A service message can be useful for request/reply interaction between node and it's composed by two messages: a request message and a response message. Every node can invoke these message to request a particular data or a particular task.

Here a short list of the benefits of using ROS in LINarm++ Manager application:

- Easy modularity and scalability of the entire system
- Easy data exchange and availability among nodes
- Good performance thanks to the multi process management

(All these feature make Linarm++ Manager an ideal software to be developed under ROS.)

As depicted in Fig. 1 the messages coming/start to/from the Linarm++ Manager module through different kind of channels. In particular, the communication between the Patient model device and the VR Rendere device is achieved by an UDP channel, while the communication between Linarm, RehabStim and Physiological Sensors device is realized by an USB channel. Hence the need to transform the message sent with this type of protocol to ROS message.

To explain how this problem has been faced, Fig. 3 can be useful.

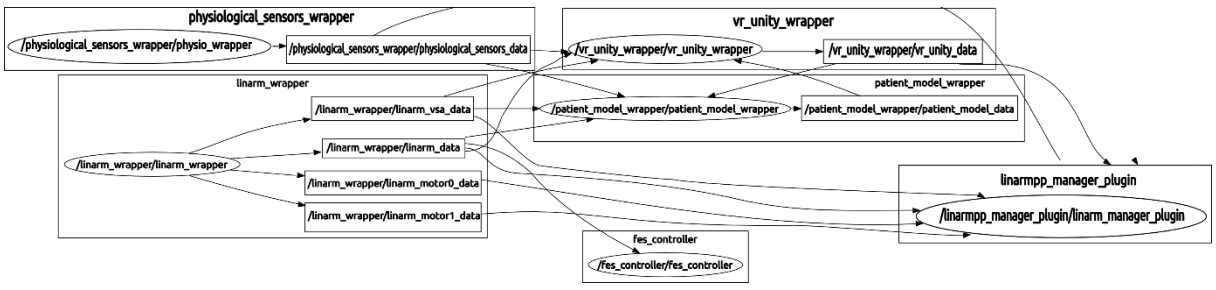


Fig. 3: ROS graph for the Linarm++ Manager software

For each module represented in Fig. 1 there is a corresponding wrapper or controller node in Fig. 3. These nodes were created to translate the message coming from UDP or USB protocol in ROS message type. Every wrapper in Fig. 3 is composed by a node, represented by the elliptic element, able to send messages to one or more topics, represented by the rectangular element. At the same time a node can receive messages from one or more topic. For clarity, in the figure are depicted only the topic transport system omitting to represent the service transport system.

To exploit the service messages made available from the wrapper nodes, a sort of API was created. These APIs are represented by libraries that make available a series of method and variable that abstract the access to service or topic. These API are used for example in the *linarmpp_manager_plugin* that provides to the user a GUI useful to program and control the Linarm++ system behavior.
linarmpp_manager_plugin has been developed for the control of the entire system. But in the Linarm++ Manager software other "manager_plugin" nodes have been developed to provide the user the control of every single module in Linarm++ system. For modules as Physiological sensors, Patient Model or VR Rendered, modules that don't provide an effective control but are only able to send data to the system, only a viewer node was considered. These viewer nodes provide only a GUI to show the data to the user. In Fig. 3 they are not reported for shortness and clarity. All these plugins are developed exploiting the *rqt* framework. It is a framework available in ROS really useful in the creation of GUI with good performance and modularity. In the sequel of this document all the plugin available in the software will be shown and explained.
Other libraries used in the development of the Linarm++ Manager software are:

- Python struct library was used to interface the wrapper with the UDP communication protol and in particular to pack and unpack the message
- Python socket library was used for the UDP communication
- Python pyserial was used to interface the wrapper with serial communication protocol
- ROS and rqt are already been widely show previously

3 Nodes of the architecture

In this chapter all the "manager_plugin" and "viewer" nodes are shown. Both the GUI interface and the software implementation are explained hereafter.

3.1 Linarm (robot) manager

The Linarm manager plugin was created to provide a simple GUI for the Linarm device control by the user. This plugin was thought with the Linarm viewer plugin that provides a GUI for the visualization of the device state. In Fig. 4 there is a representation of the Linarm manager and viewer.

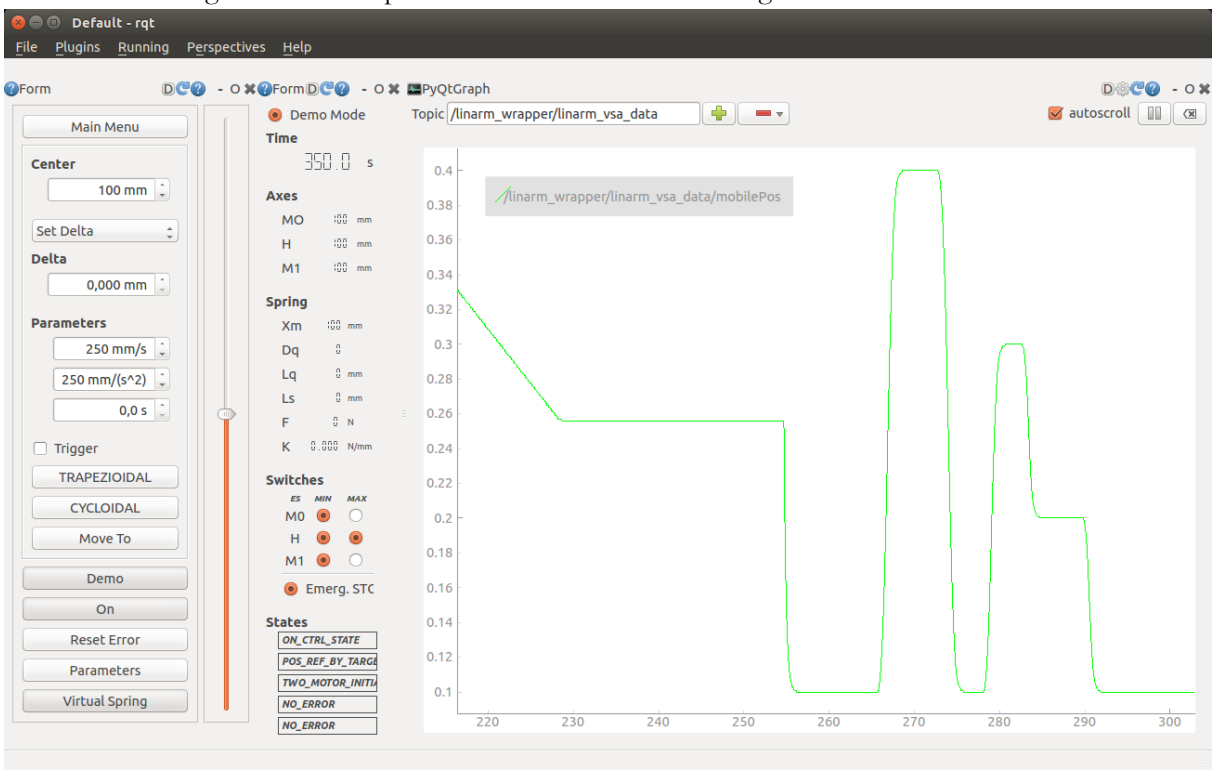


Fig. 4: Linarm manager plugin and ROS graph

The left-most block, represents the Linarm manager plugin. This plugin is provided by a series of button and line input useful to control the state, the modality, the position, the admittance and assistance parameter and many others properties of the robot. In the viewer all of this quantities and value are depicted in an easy-to-see interface.

Fig. 4 represents also a graph for some quantities. It is important to remark that this plugin is provided by the rqt framework and has not specifically been created for the Linarm++ scope.

By the Linarm manager plugin it's also possible write some script useful to program the robot for performing the desired task Fig. 5. It's important to clarify that this plugin, and therefore also the script modality, is completely blind w.r.t. the other modules of the Linarm system.

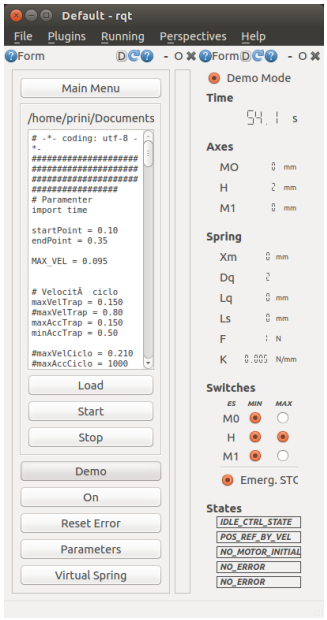


Fig. 5: Linarm manager GUI in script mode

Fig. 6 summarizes the configuration of the node. The Linarm manager plugin nodes, that implements the Linarm manager API, can communicate through topic and service transport system with the Linarm wrapper. Exploiting these channels, the Linarm manager plugin can request particular tasks or parameters.

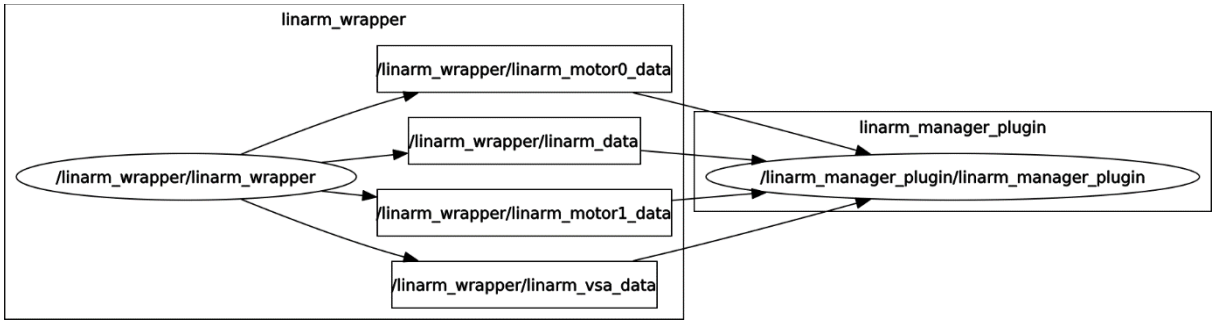


Fig. 6: Linarm manager ROS graph

3.2 Physiological sensors viewer

As previously said some module, as the physiological sensors module, do not require particular command for the control. For example the Physiological sensors module have the only function of sampling the physiological sensors and send the data to the Linarm++ system. For this reason the Physiological sensors plugin was though only for a data visualization and not for control. The Physiological sensors viewer plugin is represented in Fig. 7.

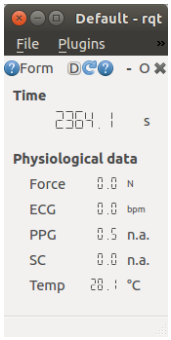


Fig. 7: Physiological sensor viewer plugin

The implementation scheme is depicted in figure 7.

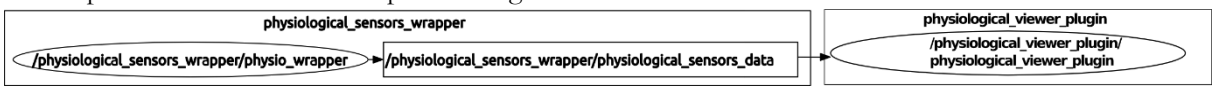


Fig. 8: Implementation scheme for physiological sensors

3.3 Patient Model viewer

As for the physiological sensors module, also the Patient model module doesn't requires special commands. Physiological sensors module only receives data, processes this data and, afterwards, sends the result (e.g. assistance parameter). For this reason, even in the patient model case, only a viewer plugin was considered. This is represented in Fig. 9.

Fig. 9: Patient model viewer plugin

On the implementation side, the connection between plugin and wrapper is similar to the scheme reported in Fig. 8. The only difference is represented by the data coming through topic channels, collected in the Patient Model wrapper. These data are collected, packed and after sent by UDP channel to the Patient Model module.

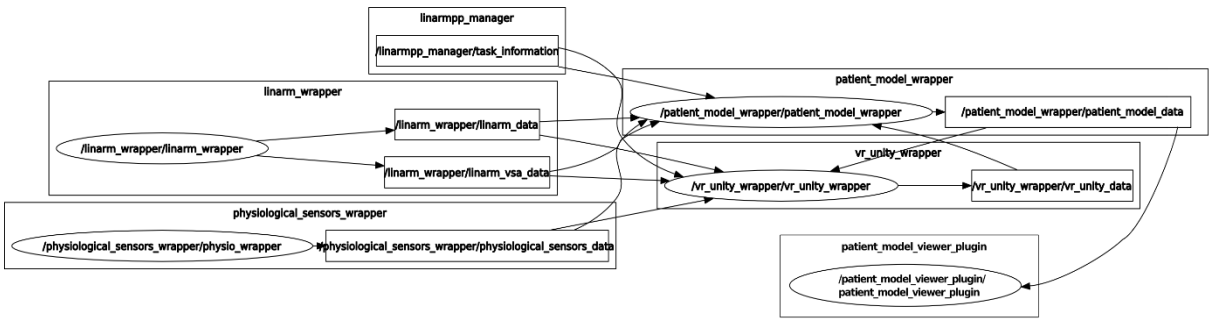


Fig. 10: Patient model nodes implementation scheme

3.4 VR Rendered viewer

The plugin is represented by a viewer plugin. The wrapper collects data from different topics, are packed and sent to the module.

Fig. 11 and Fig. 12 respectively represent the VR Rendered viewer plugin and the implementation scheme.

Fig. 11: VR Rendered viewer plugin

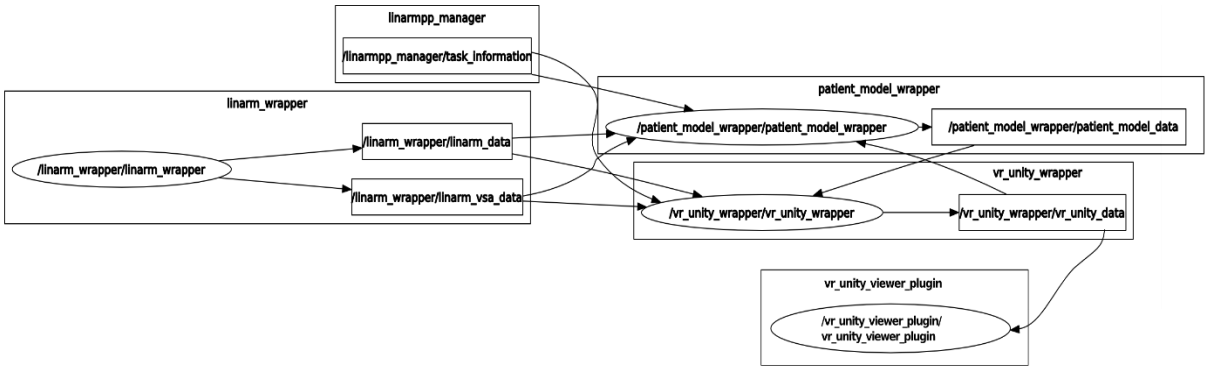


Fig. 12: VR Rendered implementation scheme

3.5 FES Manager Plugin

As reported in D5.2, the stimulator chosen to realize the FES module is Rehastim. This stimulator can be controlled by USB thanks to the ScienceMode stimulator function. A python library has been developed to control the stimulator. Afterwards, a ROS wrapper node coded in python was realized, to make the FES stimulator communicate within the ROS-based architecture.

For a correct use of the stimulator it is important that the muscles are correctly activated during the movement. The example of a stimulation current profile within the movement of a frontal reaching movement is represented in Fig. 13.

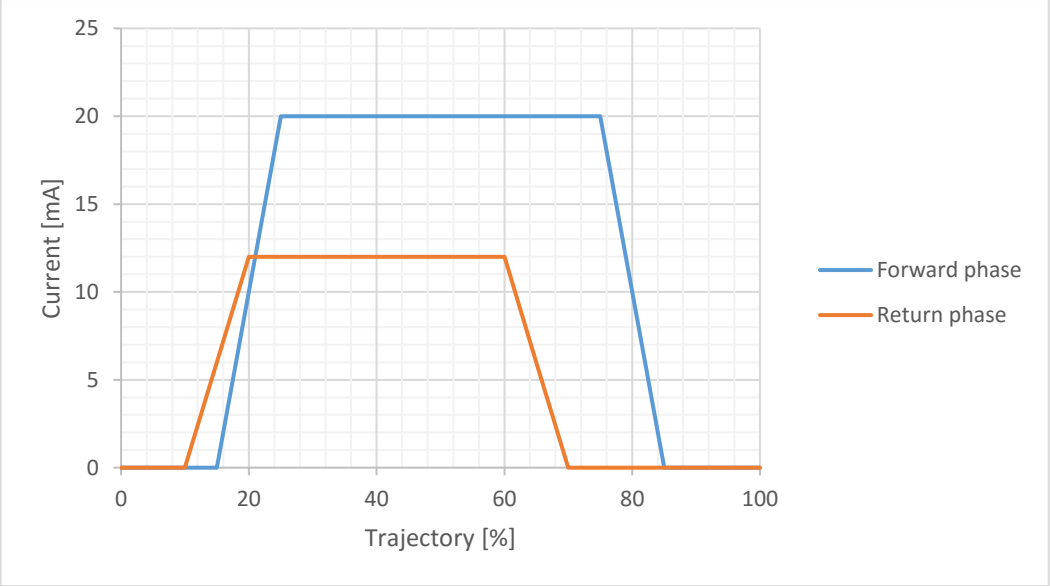


Fig. 13: Chart stimulation profile example

Thanks to the FES wrapper node and through a particular service is possible to send to the wrapper node a particular current and pulse width profile. After this, when a start command is sent to the stimulator, the FES wrapper node adapt the current or pulse width level of the stimulator to those indicated by the medical personnel.

For the control and for the profile definition a FES manager plugin was realized. The FES manager consists of two main dialog windows. The first, reported in Fig. 14, is useful to start and stop the stimulation and to load a particular profile. Furthermore through this interface the medical personnel is able to change current and pulse width value online thanks to two virtual knobs.

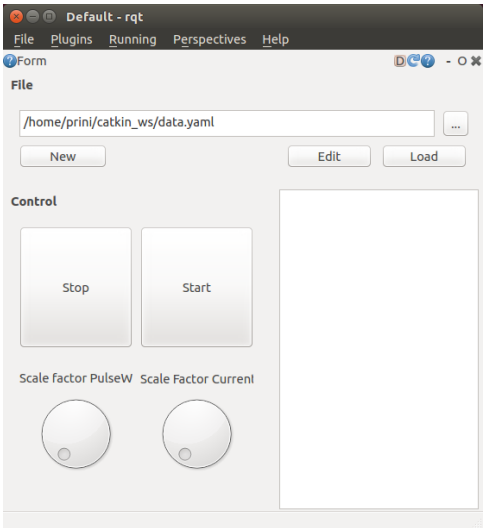


Fig. 14: FES manager plugin interface

The second window (Fig. 15) is useful for the profile definition. In this window there are basically two table useful to the definition of the trapezoidal profile for current and pulse width.

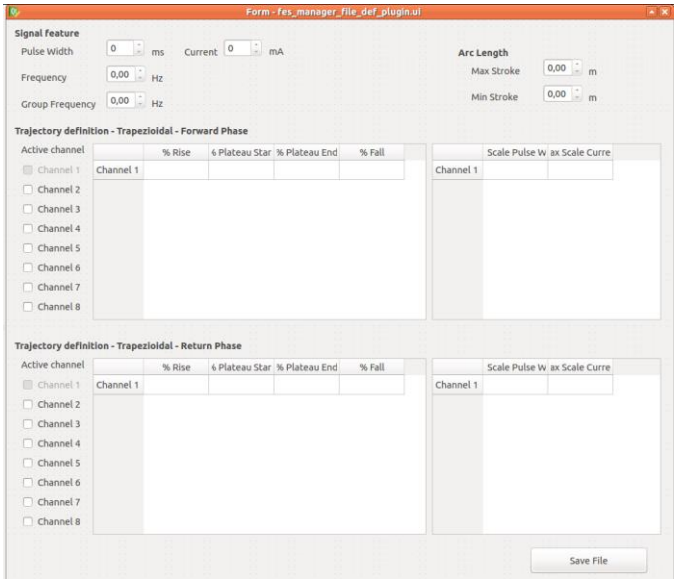


Fig. 15: FES exercise definition interface

The implementation scheme follows the same logic of the other scheme. The manager plugin communicates through services and topics with the wrapper that manage the stimulator.

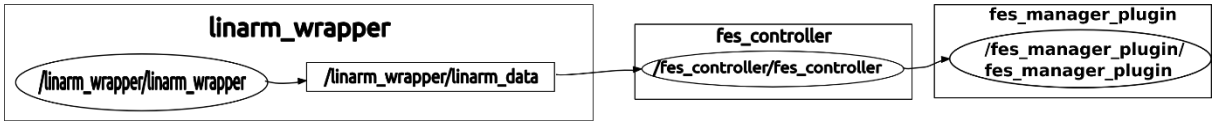


Fig. 16: Implementation scheme for FES manager plugin

3.6 Linarm++ Manager Plugin

This last but not least plugin is the one that permits the connection among all the nodes. Almost all the API functionalities of the entire system are included in this plugin. In this way it is possible to command all the system through some simple scripts. For example, it is possible to link or compare different data coming from the modules of the system and take decisions on the base of results.
The GUI is depicted in Fig. 17. It is very simple and comprise only a text editor for the script and start and stop buttons. All the usable methods and variables are reported in the manual reported in the Appendix section.

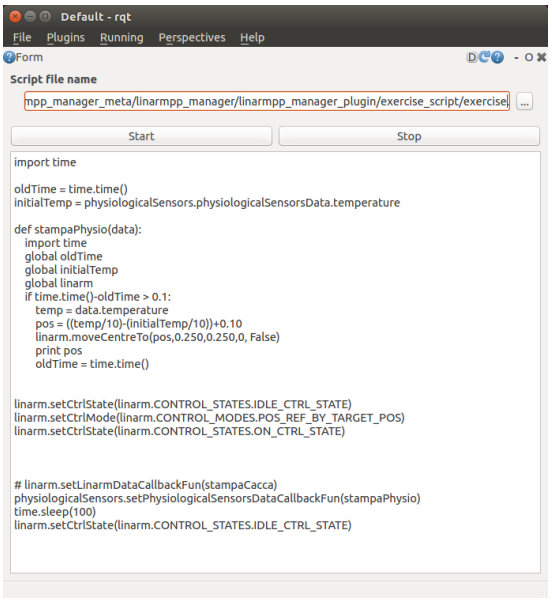


Fig. 17: Linarm++ manager plugin interface

4 Simulation

This simulation have the purpose of presenting the potentiality of this system of connecting all the nodes to determine the behavior of the overall LINarm++ platform.

It simulate a possible connection between the physiological sensors signal and the LINarm mechatronic device. In this example the patient is subjected to a passive exercise by the LINarm robot, with the patient's arm being moved by the robot. The forward phase and return phase cycle is repeated four times. In this exercise each movement is triggered by the grasp, which has to reach a predefined threshold value to start the movement.

As mentioned before, the only plugin software able to use all the Linarm++ module is the Linarm++ Manager plugin. This paradigmatic exercise is implemented taking the most of the Python language enabling to encode exercise by scripts.

The script for this particular exercise is reported in Fig. 18.

```
# -*- coding: utf-8 -*-
################################################################################
# Paramenter
import time
startPoint = 0.10
endPoint = 0.40

# Velocità ciclo
maxVelTrap = 0.150
maxAccTrap = 0.150
minAccTrap = 0.50

numCicli = 4

DeltaPos1 = 0.010

forceThreshold = 30
################################################################################

def waitCondition(cond, loc, timeoutReaching):
    import time
    old = time.time()
    while (eval(cond, globals(), loc) == False and (time.time()- old < timeoutReaching)):
        pass

linarm.setCtrlState(linarm.CONTROL_STATES.IDLE_CTRL_STATE)
linarm.setCtrlMode(linarm.CONTROL_MODES.POS_REF_BY_TARGET_POS)
linarm.setCtrlState(linarm.CONTROL_STATES.ON_CTRL_STATE)

linarm.moveDeltaTo(DeltaPos1,maxVelTrap,maxAccTrap,0, False)


for i in range(0,numCicli):
    waitCondition("physiologicalSensors.physiologicalSensorsData.force>20", locals(), 10)
    linarm.moveCentreTo(endPoint,maxVelTrap,maxAccTrap,0, False)
    waitCondition("abs(endPoint-linarm.linarmVsaState.equilibriumPos)<5e-3", locals(), 10)
    waitCondition("physiologicalSensors.physiologicalSensorsData.force>20", locals(), 10)
    linarm.moveCentreTo(startPoint,maxVelTrap,maxAccTrap,0, False)
    waitCondition("abs(startPoint-linarm.linarmVsaState.equilibriumPos)<5e-3", locals(), 10)

linarm.setCtrlState(linarm.CONTROL_STATES.IDLE_CTRL_STATE)
```

Fig. 18: trigger exercise code

In this script two object are used; the *linarm* object and the *physiologicalSensor* object. A *waitCondition* function enables the triggered behavior as implemented in line 34 and 37 of Fig. 18.
The robot behavior is reported in Fig. 19. The left graph reports the grasp force, while the right graph reports the position of the robot.

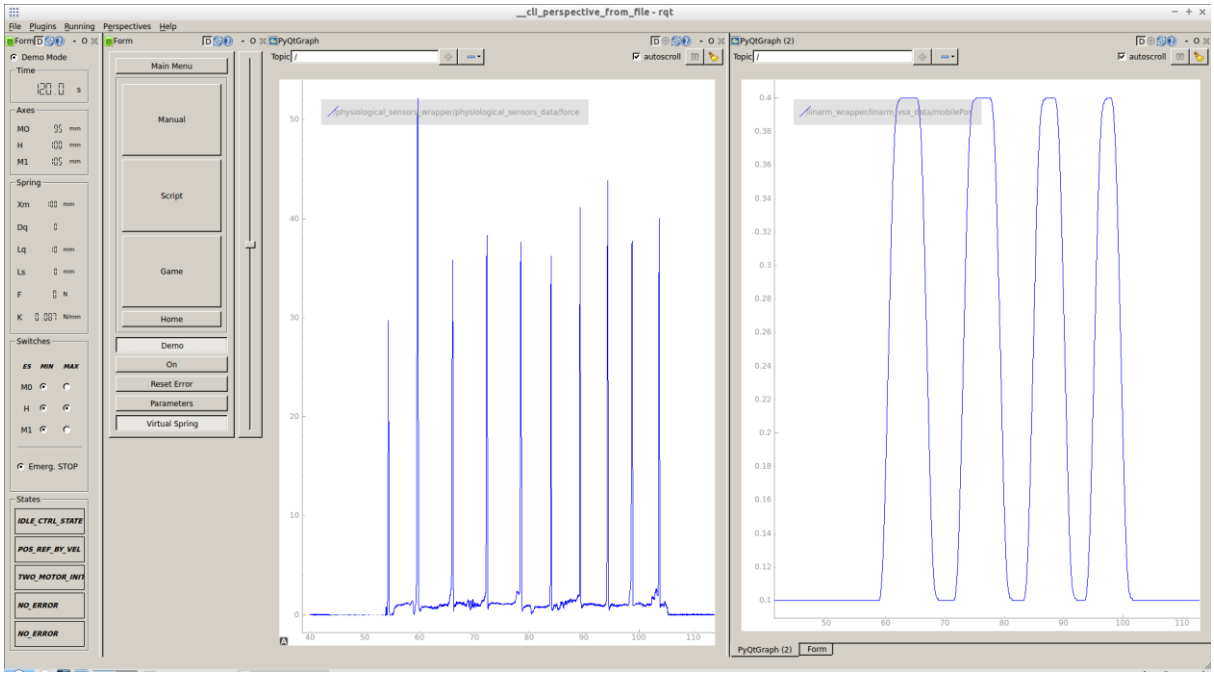


Fig. 19: Trigger force exercise result

It is important to see that only if the threshold force value is exceeded the movement start.
All the class and relative methods a variable, available for this type of script are reported in the Appendix.

Appendix

In the next pages the documentation of the LINarm++ Manager is reported.

Linarmpp manager package

0.0.1

Generated by Doxygen 1.8.6

Fri Jul 29 2016 23:43:34

Contents

Chapter 1

Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Namespace Documentation

4.1 fes_user Namespace Reference

This package was created to provide a complete support for the manage of the FES module in the linarm++ system platform.

4.1.1 Detailed Description

This package was created to provide a complete support for the manage of the FES module in the linarm++ system platform.

4.2 linarm_user Namespace Reference

This package was created to provide a support in the Linarm2 robot control.

4.2.1 Detailed Description

This package was created to provide a support in the Linarm2 robot control. This package contain function and class useful as wrapper from the ROS service and a useful python language.

4.3 patient_model_user Namespace Reference

This package was created to provide a support to manage the patient model data in the linarm++ software platform.

4.3.1 Detailed Description

This package was created to provide a support to manage the patient model data in the linarm++ software platform.

4.4 physiological_sensors_user Namespace Reference

This package was created to provide a support to manage the physiological sensors data in the linarm++ software platform.

4.4.1 Detailed Description

This package was created to provide a support to manage the physiological sensors data in the linarm++ software platform.

4.5 vr_unity_user Namespace Reference

This package was created to provide a support to manage the vr unity data in the linarm++ software platform.

4.5.1 Detailed Description

This package was created to provide a support to manage the vr unity data in the linarm++ software platform.

Chapter 5

Class Documentation

5.1 linarmpp_manager_pkg.fes_user.FESUser Class Reference

FES User class.

Public Member Functions

- def __init__

  Constructor.
- def **getConfigData**
- def loadFile

  Load File.
- def saveFile

  Save File.

Public Attributes

- **classString**
- start

  Start.
- stop

  Stop.
- setPWScaleFactor

  Set PulseWidth scale factor.
- setCurrentScaleFactor

  Set Current scale factor.

5.1.1 Detailed Description

FES User class.
this class was created to provide a complete support to manage the RehaStim stimulator

```
FES user class

FESUser is a class able to manage all the message and service from/to FES module wrapper. The wrapper convert
```

5.1.2 Constructor & Destructor Documentation

5.1.2.1 def linarmpp_manager_pkg.fes_user.FESUser.__init__ (*self*)

Constructor.

Verify the presence of the Patient model wrappe

5.1.3 Member Function Documentation

5.1.3.1 def linarmpp_manager_pkg.fes_user.FESUser.loadFile (*self, fileName*)

Load File.

Useful to load current and pulsewidth profile in the wrapper. The profile must be saved in yaml file.

Parameters

in	*fileName(str)*	Name of the stimulator profile file

5.1.3.2 def linarmpp_manager_pkg.fes_user.FESUser.saveFile (*self, data, fileName*)

Save File.

Useful to save the current and pulsewidth profile in file. You to provide the data in a load_paramRequest structure type.

Parameters

in	*data(load_paramRequest)*	profile data
in	*fileName(str)*	Name of the saved file

5.1.4 Member Data Documentation

5.1.4.1 linarmpp_manager_pkg.fes_user.FESUser.setCurrentScaleFactor

Set Current scale factor.

Set a scale factor for the current profile. Useful to test online the acceptance level for the patient

5.1.4.2 linarmpp_manager_pkg.fes_user.FESUser.setPWScaleFactor

Set PulseWidth scale factor.

Set a scale factor for the pulsewidth profile. Useful to test online the acceptance level for the patient

5.1.4.3 linarmpp_manager_pkg.fes_user.FESUser.start

Start.

start the stimulator with the loaded profile

5.1.4.4 linarmpp_manager_pkg.fes_user.FESUser.stop

Stop.

stop the stimulator.

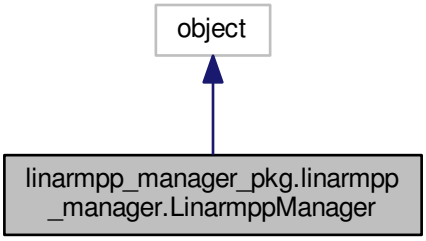
The documentation for this class was generated from the following file:

- src/linarmpp_manager_pkg/fes_user.py

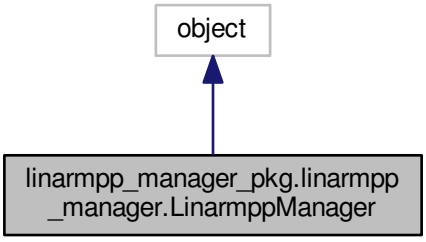
5.2 linarmpp_manager_pkg.linarmpp_manager.LinarmppManager Class Reference

Linarm++ manager class.

Inheritance diagram for linarmpp_manager_pkg.linarmpp_manager.LinarmppManager:



Collaboration diagram for linarmpp_manager_pkg.linarmpp_manager.LinarmppManager:



Public Member Functions

- def **__init__**
- def **internalExec**
- def **setProgram**
- def **startProgram**

Public Attributes

- **classString**
- **fileName**

- **linarm**
- **physiologicalSensors**
- **VRUnity**
- **patientModel**

5.2.1 Detailed Description

Linarm++ manager class.

This class provide an access to all the funzionality of the linarm++ system. Trought this class is possible manage and update this module:

- linarm robot
- physiological sensors module
- patient model module
- vr unity module
- FES module TODO

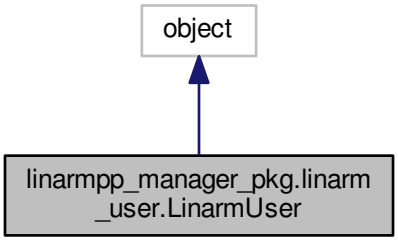
The documentation for this class was generated from the following file:

- src/linarmpp_manager_pkg/linarmpp_manager.py

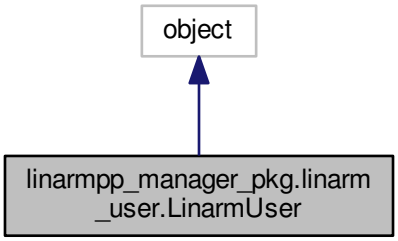
5.3 linarmpp_manager_pkg.linarm_user.LinarmUser Class Reference

Linarm User class.

Inheritance diagram for linarmpp_manager_pkg.linarm_user.LinarmUser:



Collaboration diagram for linarmpp_manager_pkg.linarm_user.LinarmUser:



Public Member Functions

- def __init__

  The constructor of LinarmUser class.
- def **initializedEnum**
- def initializeTopic

  Useful to subscribe to all the Linarm ROS topic.
- def linarmDataCallback

  Callback for Linarm Data update.
- def linarmMotor0DataCallback

  Callback for Linarm motor0 Data update.
- def linarmMotor1DataCallback

  Callback for Linarm motor1 Data update.
- def linarmVsaDataCallback

  Callback for Linarm VSA Data update.
- def setLinarmDataCallbackFun

  Useful to set a list of callback in LinarmData update.
- def setLinarmMotor0DataCallbackFun

  Useful to set a list of callback in LinarmMotor0Data update.
- def setLinarmMotor1DataCallbackFun

  Useful to set a list of callback in Linarmmotor1Data update.
- def setLinarmVsaDataCallbackFun

  Useful to set a list of callback in LinarmVSAData update.
- def initializeService

  This function create all the wrapper between ROS service and python function.
- def **getConfigData**

Public Attributes

- **classString**
- **CONTROL_STATES**
- **CONTROL_MODES**
- **HOME_STATES**
- **ERROR_STATES**
- **TRIGGER_MODES**

- **linarmDataCallbackList**
- **linarmMotor0DataCallbackList**
- **linarmMotor1DataCallbackList**
- **linarmVsaDataCallbackList**
- **linarmState**
- **linarmMotor0State**
- **linarmMotor1State**
- **linarmVsaState**
- setDemo

  Set Demo.
- resetError

  Reset error.
- setCtrlState

  Set Control state.
- setCtrlMode

  Set Control mode.
- setCentreVelocity

  Set centre velocity.
- setDeltaVelocity

  Set delta velocity.
- getPidTunings

  Get PID tunings.
- moveCentreTo

  Move centre to.
- moveDeltaTo

  Move delta to.
- setStiffness

  Set stiffness.
- goToHome

  Go to home.
- setTrajectoryType

  Set trajectory type.
- setAdmittanceParameter

  Set admittance parameter.
- getAdmittanceParameter

  Get admittance parameter.
- getAssistanceParameter

  Set assistance parameter.
- setAssistanceParameter

  Get assistance parameter.
- setAssistanceTargetPos

  Set assistance target pos.
- getAssistanceTargetPos

  Get assistance target pos.
- enableVirtualVsa

  Enable/Disable virtual spring.
- setVirtualVsaSpringDisplacement

  Set virtual VSA spring displacement.
- **getIsInitialized**
- disableEndstrokes

  Disable endstroke.

- setTriggerMode

  Set trigger mode [NOT USED].
- setTriggerParameter

  Set trigger parameter [NOT USED].

5.3.1 Detailed Description

Linarm User class.

LinarmUser class is a class able to control the linarm robot. Not all the robot ROS service are available in this class.

```
LinarmUser class

LinarmUser class is a class able to control the linarm robot. Not all the robot service are available in this
List of service available:
- /linarm_wrapper/set_demo -> setDemo
- /linarm_wrapper/reset_error -> resetError
- /linarm_wrapper/set_control_state -> setCtrlState
- /linarm_wrapper/set_control_mode -> setCtrlMode
- /linarm_wrapper/set_centre_velocity -> setCentreVelocity
- /linarm_wrapper/set_delta_velocity -> setDeltaVelocity
- /linarm_wrapper/get_pid_tunings -> getPidTunings
- /linarm_wrapper/move_centre_to -> moveCentreTo
- /linarm_wrapper/move_delta_to -> moveDeltato
- /linarm_wrapper/set_stiffness ->setStiffness
- /linarm_wrapper/go_to_home -> goToHome
- /linarm_wrapper/set_trajectory_type -> setTrajectoryType
- /linarm_wrapper/set_assistance_target_pos -> setAssistanceTargetPos
- /linarm_wrapper/get_assistance_target_pos -> getAssistanceTargetPos
- /linarm_wrapper/enable_virtual_vsa -> enableVirtualVsa
- /linarm_wrapper/set_virtual_vsa_displacement -> setVirtualVsaSpringDisplacement
- /linarm_wrapper/get_initialization_state -> getIsInitialized
- /linarm_wrapper/disable_endstroke -> disableEndstrokes
- /linarm_wrapper/set_trigger_mode -> setTriggerMode
- /linarm_wrapper/set_trigger_parameter -> setTriggerParameter
List of not available service:
- /linarm_wrapper/start_data_stream -> startDataStream
- /linarm_wrapper/stop_data_stream -> stopDataStream
- /linarm_wrapper/set_motor_offset -> setMotorOffset
- /linarm_wrapper/set_mobile_offset -> setMobileOffset
- /linarm_wrapper/set_motor_ratio -> setMotorRatio
- /linarm_wrapper/set_mobile_ratio -> setMobileRatio
- /linarm_wrapper/set_max_tracking_error -> setMaxTrackingError
- /linarm_wrapper/set_max_force -> setMaxForce
- /linarm_wrapper/set_max_current -> setMaxCurrent
- /linarm_wrapper/set_max_acceleration -> setMaxAcceleration
- /linarm_wrapper/set_max_velocity -> setMaxVelocity
- /linarm_wrapper/set_home_prameter -> setHomeParameter
- /linarm_wrapper/set_position_limit -> setPosLimit
- /linarm_wrapper/set_pid_tunings -> setPidTunings
```

5.3.2 Member Function Documentation

5.3.2.1 def linarmpp_manager_pkg.linarm_user.LinarmUser.initializeService (*self*)

This function create all the wrapper between ROS service and python function.

Here is represent the list of service available:

- /linarm_wrapper/set_demo -> setDemo
- /linarm_wrapper/reset_error -> resetError
- /linarm_wrapper/set_control_state -> setCtrlState
- /linarm_wrapper/set_control_mode -> setCtrlMode

- /linarm_wrapper/set_centre_velocity -> setCentreVelocity
- /linarm_wrapper/set_delta_velocity -> setDeltaVelocity
- /linarm_wrapper/get_pid_tunings -> getPidTunings
- /linarm_wrapper/move_centre_to -> moveCentreTo
- /linarm_wrapper/move_delta_to -> moveDeltato
- /linarm_wrapper/set_stiffness ->setStiffness
- /linarm_wrapper/go_to_home -> goToHome
- /linarm_wrapper/set_trajectory_type -> setTrajectoryType
- /linarm_wrapper/set_assistance_target_pos -> setAssistanceTargetPos
- /linarm_wrapper/get_assistance_target_pos -> getAssistanceTargetPos
- /linarm_wrapper/enable_virtual_vsa -> enableVirtualVsa
- /linarm_wrapper/set_virtual_vsa_displacement -> setVirtualVsaSpringDisplacement
- /linarm_wrapper/get_initialization_state -> getIsInitialized
- /linarm_wrapper/disable_endstroke -> disableEndstrokes
- /linarm_wrapper/set_trigger_mode -> setTriggerMode
- /linarm_wrapper/set_trigger_parameter -> setTriggerParameter

5.3.3 Member Data Documentation

5.3.3.1 linarmpp_manager_pkg.linarm_user.LinarmUser.disableEndstrokes

Disable endstroke.

To disable switch on end stroke effect.

5.3.3.2 linarmpp_manager_pkg.linarm_user.LinarmUser.enableVirtualVsa

Enable/Disable virtual spring.

This function enable or disable the virtual spring. Virtual spring is useful in demo mode without the presence of the robot for someany kind of simulation.

Parameters

in	*virtual*	bool Boolean to enable or disable the vrtual spring

5.3.3.3 linarmpp_manager_pkg.linarm_user.LinarmUser.getAdmittanceParameter

Get admittance parameter.

Useful to get the admittance parameter in POS_REF_BY_ADMITTANCE and POS_REF_BY_ASSISTANCE mode. Admittance parameter is used to set the velocity function of the force measured.

Parameters

out	*Ka*	float Admittance parameter

5.3.3.4 linarmpp_manager_pkg.linarm_user.LinarmUser.getAssistanceParameter

Set assistance parameter.

Useful to set the assistance parameter in POS_REF_BY_ASSISTANCE mode. Assistance parameter is used to set stiffness of the assistance virtual spring.

Parameters

in	*Kf*	float Assistance parameter

5.3.3.5 linarmpp_manager_pkg.linarm_user.LinarmUser.getAssistanceTargetPos

Get assistance target pos.

Useful to get the assistance virtual spring target position in POS_REF_BY_ASSISTANCE mode. Target position represent a sort of origin for the assistance virtual spring.

Parameters

out	*Apos*	float Assistance target position

5.3.3.6 linarmpp_manager_pkg.linarm_user.LinarmUser.getPidTunings

Get PID tunings.

Return PID parameters of the linarm motors control.

Parameters

in	*index*	int index of the motor
out	*PIDParam*	tuple A tuple with Proportional, Integrative, Derivative parameters

5.3.3.7 linarmpp_manager_pkg.linarm_user.LinarmUser.goToHome

Go to home.

Active the homing procedure for the Linarm robot. If the procedure for getting zero position is already done, simply move the robot on the home position

5.3.3.8 linarmpp_manager_pkg.linarm_user.LinarmUser.moveCentreTo

Move centre to.

Set the desired position for the handle in POS_REF_BY_TARGET_POS mode

Parameters

in	*targetPos*	float the desired position in meter
in	*maxVel*	float maximum velocity reached in the movement

in	*maxAcc*	float maximum acceleration reached in the movement
in	*execTime*	float desired duration of the movement. If 0 the trajectory is generated on the maxVel and maxAcc parameter
in	*trigger*	bool always False

5.3.3.9 linarmpp_manager_pkg.linarm_user.LinarmUser.moveDeltaTo

Move delta to.

Set the desired position for the ends of VSA spring IN POS_REF_BY_TARGET_POS mode

Parameters

in	*targetDeltaPos*	float the desired position in meter
in	*maxVel*	float maximum velocity reached in the movement
in	*maxAcc*	float maximum acceleration reached in the movement
in	*execTime*	float desired duration of the movement. If 0 the trajectory is generated on the maxVel and maxAcc parameter
in	*trigger*	bool always False

5.3.3.10 linarmpp_manager_pkg.linarm_user.LinarmUser.resetError

Reset error.

Reset the error state and bring the robot to a NO_ERROR state

5.3.3.11 linarmpp_manager_pkg.linarm_user.LinarmUser.setAdmittanceParameter

Set admittance parameter.

Useful to set the admittance parameter in POS_REF_BY_ADMITTANCE and POS_REF_BY_ASSISTANCE mode. Admittance parameter is used to set the velocity function of the force measured.

Parameters

in	*Ka*	float Admittance parameter

5.3.3.12 linarmpp_manager_pkg.linarm_user.LinarmUser.setAssistanceParameter

Get assistance parameter.

Useful to get the assistance parameter in POS_REF_BY_ADMITTANCE mode. Assistance parameter is used to set stiffness of the assistance virtual spring.

Parameters

out	*Ka*	float Admittance parameter

5.3.3.13 linarmpp_manager_pkg.linarm_user.LinarmUser.setAssistanceTargetPos

Set assistance target pos.

Useful to set the assistance virtual spring target position in POS_REF_BY_ASSISTANCE mode. Target position represent a sort of origin for the assistance virtual spring.

Parameters

in	*Apos*	float Assistance target position

5.3.3.14 linarmpp_manager_pkg.linarm_user.LinarmUser.setCentreVelocity

Set centre velocity.

Set the centre velocity of the handle in POS_REF_BY_VEL mode.

Parameters

in	*centreVelocity*	float desired velocity of the equilibrium position of the handle

5.3.3.15 linarmpp_manager_pkg.linarm_user.LinarmUser.setCtrlMode

Set Control mode.

Set the robot control mode.

Parameters

in	*controlMode(int)*	desired control mode example... controlMode = 0 -> POS_REF_BY_VEL controlMode = 1 -> POS_REF_BY_TARGET_POS controlMode = 2 -> POS_REF_BY_IMPEDANCE controlMode = 3 -> POS_REF_BY_ADMITTANCE controlMode = 4 -> POS_REF_BY_ASSISTANCE

5.3.3.16 linarmpp_manager_pkg.linarm_user.LinarmUser.setCtrlState

Set Control state.

Set the robot control state.

Parameters

in	*controlState(int)*	desired control state example... controlState = 0 -> IDLE_CTRL_STATE controlState = 1 -> HOME_CTRL_STATE controlState = 2 -> ON_CTRL_STATE

5.3.3.17 linarmpp_manager_pkg.linarm_user.LinarmUser.setDeltaVelocity

Set delta velocity.

Set the relative velocity of the VSA spring ends in POS_REF_BY_VEL mode.

Parameters

in	*deltaVelocity*	float desired velocity of the delta for the VSA spring

5.3.3.18 linarmpp_manager_pkg.linarm_user.LinarmUser.setDemo

Set Demo.

Parameters

in	*demo*	bool demo state

5.3.3.19 linarmpp_manager_pkg.linarm_user.LinarmUser.setStiffness

Set stiffness.

Set the desired position for the ends of VSA spring IN POS_REF_BY_TARGET_POS mode to reach the desidered stiffness from the VSA.

Parameters

in	*stiffness*	float desired stiffness in Newton/meter
in	*maxVel*	float maximum velocity reached in the movement
in	*maxAcc*	float maximum acceleration reached in the movement
in	*execTime*	float desired duration of the movement. If 0 the trajectory is generated on the maxVel and maxAcc parameter

5.3.3.20 linarmpp_manager_pkg.linarm_user.LinarmUser.setTrajectoryType

Set trajectory type.

Funtion to select the desired trajectory shape. TRAPEZIOIDAL and CYCLOIDAL shape are available.

Parameters

in	*type*	int trajectory type. 0 for TRAPEZIOIDAL 1 for CYCLOIDAL

5.3.3.21 linarmpp_manager_pkg.linarm_user.LinarmUser.setTriggerMode

Set trigger mode [NOT USED].

Set the trigger mode.

Parameters

in	*mode*	int Trigger mode

5.3.3.22 linarmpp_manager_pkg.linarm_user.LinarmUser.setTriggerParameter

Set trigger parameter [NOT USED].

Set the trigger parameter.

Parameters

in	*param*	float Trigger parameter

5.3.3.23 linarmpp_manager_pkg.linarm_user.LinarmUser.setVirtualVsaSpringDisplacement

Set virtual VSA spring displacement.

This function set the virtual spring displacement with the respect to the equilibrium position.

Parameters

in	*disp*	float Virtual spring displacement

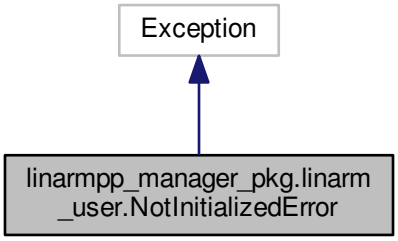
The documentation for this class was generated from the following file:

- src/linarmpp_manager_pkg/linarm_user.py

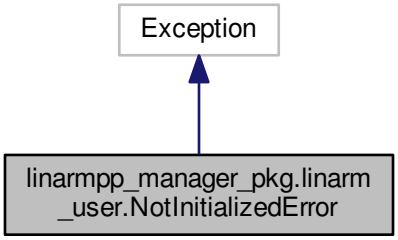
5.4 linarmpp_manager_pkg.linarm_user.NotInitializedError Class Reference

NotInitializedError class.

Inheritance diagram for linarmpp_manager_pkg.linarm_user.NotInitializedError:



Collaboration diagram for linarmpp_manager_pkg.linarm_user.NotInitializedError:



Public Member Functions

- def **__init__**

Public Attributes

- **message**

5.4.1 Detailed Description

NotInitializedError class.

It's only an exepction raised in case the linarm manager and linarm wrapper node are not turning on

```
Exception raised in case of absence for linarm manager.

Attributes:
    expression -- input expression in which the error occurred
    message -- explanation of the error
```

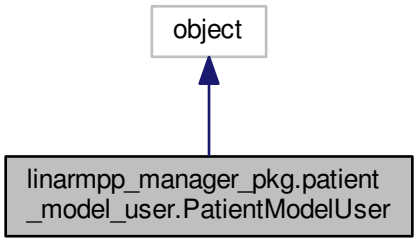
The documentation for this class was generated from the following file:

- src/linarmpp_manager_pkg/linarm_user.py

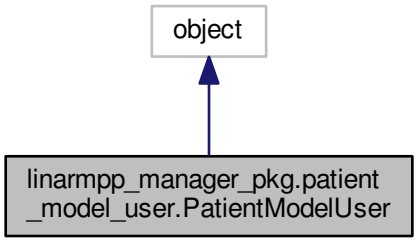
5.5 linarmpp_manager_pkg.patient_model_user.PatientModelUser Class Reference

Patient Model User class.

Inheritance diagram for linarmpp_manager_pkg.patient_model_user.PatientModelUser:



Collaboration diagram for linarmpp_manager_pkg.patient_model_user.PatientModelUser:



Public Member Functions

- def __init__

Constructor.

- def initializeTopic

  Subscribe to the topic.

- def patientModelDataCallback

  Callback function to save the data in VRUnityData and recall al the other callback function.

- def setPatientModelDataCallbackFun

  Useful to set a list of callback function on the vr unity data update.

Public Attributes

- **classString**
- **patientModelDataCallbackList**
- **patientModelData**

5.5.1 Detailed Description

Patient Model User class.
this class provide a complete support to the patient model module.

```
Patient model user class

PatientModelUser is a class able to manage the patient model data coming from the patient model module and wra
```

5.5.2 Constructor & Destructor Documentation

5.5.2.1 def linarmpp_manager_pkg.patient_model_user.PatientModelUser.__init__ (*self*)

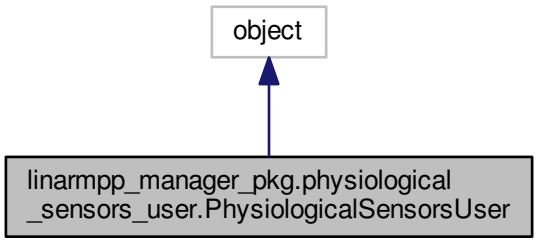
Constructor.
Verify the presence of the Patient model wrapper
The documentation for this class was generated from the following file:

- src/linarmpp_manager_pkg/patient_model_user.py

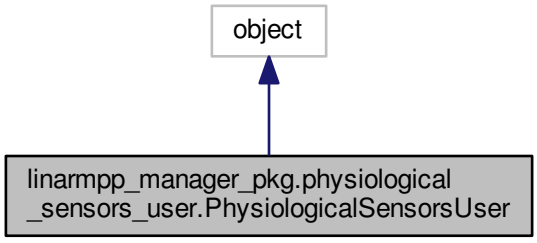
5.6 linarmpp_manager_pkg.physiological_sensors_user.PhysiologicalSensorsUser Class Reference

Physiological sensors user class.

Inheritance diagram for linarmpp_manager_pkg.physiological_sensors_user.PhysiologicalSensorsUser:



Collaboration diagram for linarmpp_manager_pkg.physiological_sensors_user.PhysiologicalSensorsUser:



Public Member Functions

- def __init__

  Constructor.
- def initializeTopic

  Subscribe to the topic.
- def physiologicalSensorsDataCallback

  Callback function to save the data in physiologicalSensorsData and recall al the other callback function.
- def setPhysiologicalSensorsDataCallbackFun

  Useful to set a list of callback function on the physiological data update.

Public Attributes

- **classString**
- **physiologicalDataCallbackList**
- **physiologicalSensorsData**

5.6.1 Detailed Description

Physiological sensors user class.
This class provide a complete support to the physiological sensors data.

```
Physiological sensors user class

Physiological sensors user is a class albe to manager the phyiological data coming from the physiological sens
```

5.6.2 Constructor & Destructor Documentation

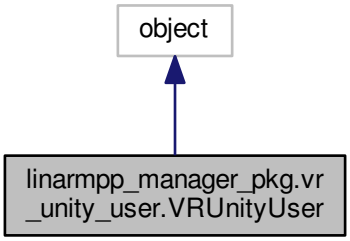
5.6.2.1 def linarmpp_manager_pkg.physiological_sensors_user.PhysiologicalSensorsUser.__init__ (*self*)

Constructor.
Verify the presence of the Physiological sensors wrapper
The documentation for this class was generated from the following file:

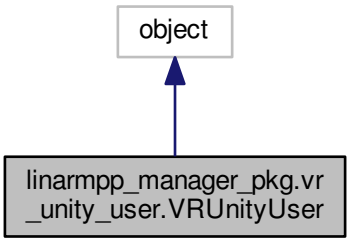
- src/linarmpp_manager_pkg/physiological_sensors_user.py

5.7 linarmpp_manager_pkg.vr_unity_user.VRUnityUser Class Reference

VR Unity user class.
Inheritance diagram for linarmpp_manager_pkg.vr_unity_user.VRUnityUser:



Collaboration diagram for linarmpp_manager_pkg.vr_unity_user.VRUnityUser:



Public Member Functions

- def __init__

  Constructor.
- def initializeTopic

  Subscribe to the topic.
- def VRUnityDataCallback

  Callback function to save the data in VRUnityData and recall al the other callback function.
- def setVRUnityDataCallbackFun

  Useful to set a list of callback function on the vr unity data update.

Public Attributes

- **classString**
- **VRUnityDataCallbackList**
- **VRUnityData**

5.7.1 Detailed Description

VR Unity user class.

This class provide a complete support to the vr unity module.

```
VR Unity user class

VRUnityUser is a class able to manage the vr_unity data coming from the vr unity module and wrapper. The wrapp
```

5.7.2 Constructor & Destructor Documentation

5.7.2.1 def linarmpp_manager_pkg.vr_unity_user.VRUnityUser.__init__ (*self*)

Constructor.

Verify the presence of the VRUnity wrapper

The documentation for this class was generated from the following file:

- src/linarmpp_manager_pkg/vr_unity_user.py

Index