

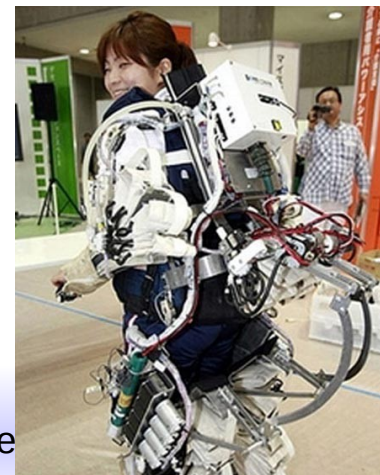
PyRehabRob

Python Graphic User Interface for Robotic Rehabilitation

(Matteo Malosio – Nicola Pedrocchi – Lorenzo Molinari Tosatti)

- Introduction to rehabilitation robotics
- Description of the prototype
- Robot control
- GUI description
- Python modules
- Conclusion and future works

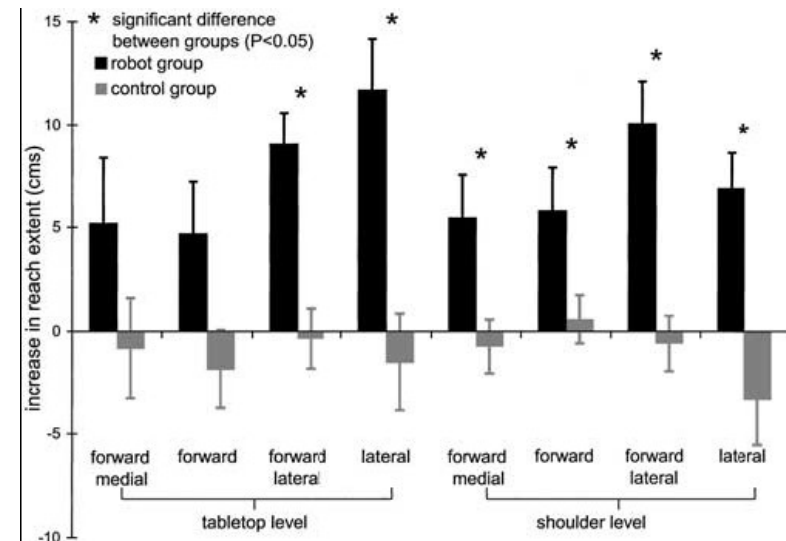
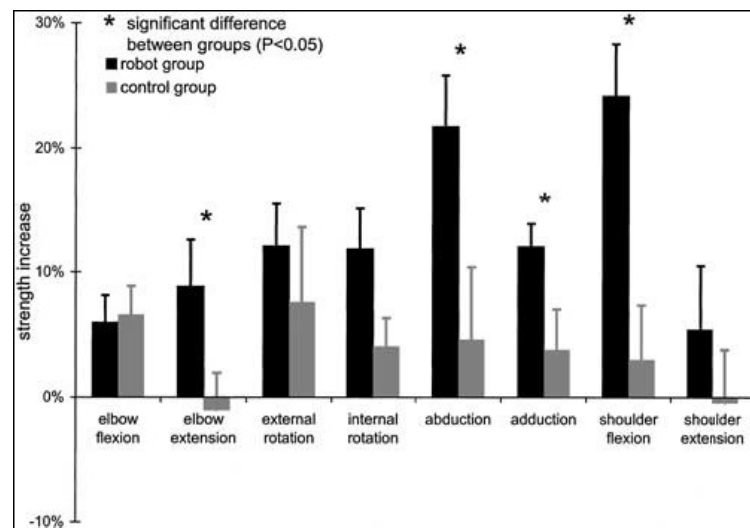
- Robot: an electro-mechanical system which conveys a sense that it has intent or agency of its own.
- A robot can:
 - move around
 - operate a mechanical limb
 - sense and manipulate their environment
 - exhibit intelligent behavior
- A robot is made up of:
 - Mechanical structure

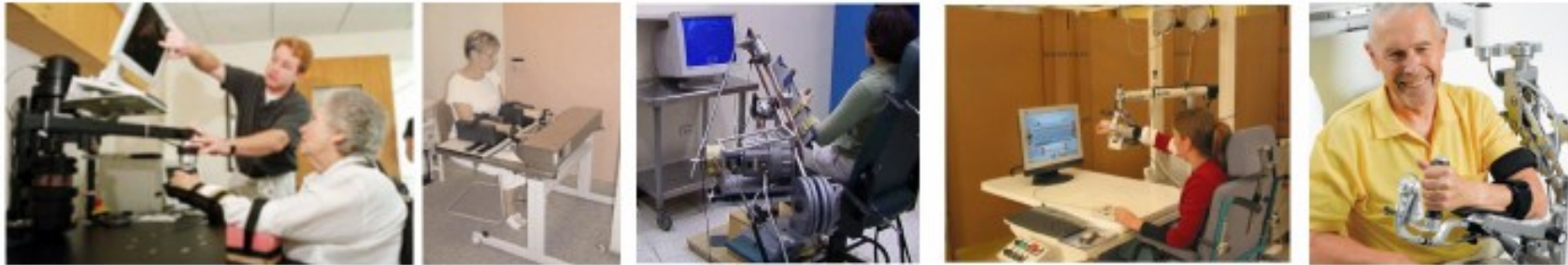


- Various possible kinematical structures
 - Serial
 - large workspace, low stiffness and dynamics
 - Parallel
 - small workspace, high stiffness and dynamics
 - Humanoid
 - service, games, social impact, mobiles



- Clinical importance
- Control algorithms can allow to achieve *passive, active and collaborative* rehabilitation
- Importance of multisensorial involvement
 - force/action-reaction, view, hear





MIT-Manus
MIT
(USA)

Bi-Manu Track
Reha-Stim
Berlin (D)

ARM-Guide
University of
California,
Irvine (USA)

GENTLE/S
FP5-EC
consortium

ARMEO
Hocoma (CH)

1990

1994

1998

2002

2006



MIME
Palo Alto
(USA)

Gait Trainer
Reha-Stim
Berlin (D)

KineAssist
Northwestern
University,
Chicago (USA)

LOKOMAT
Hocoma (CH)

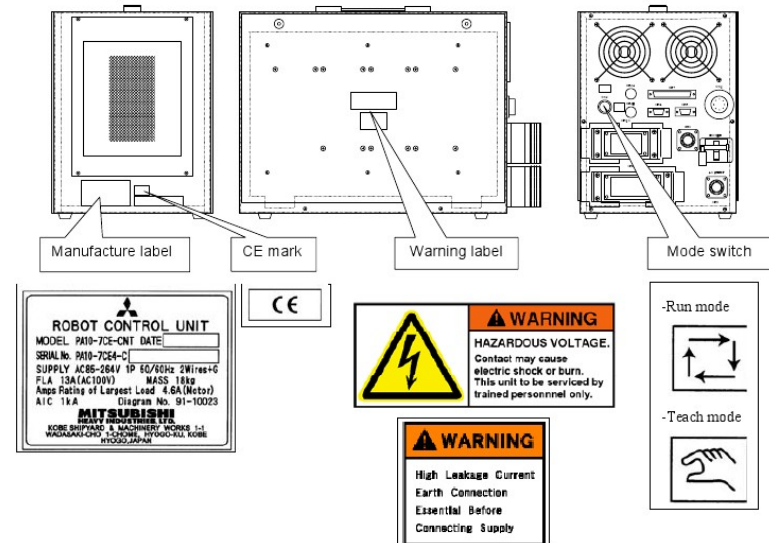
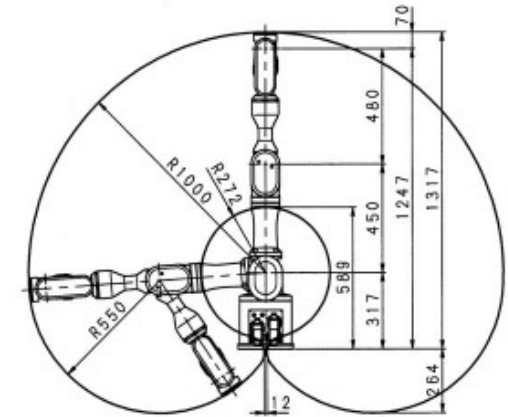
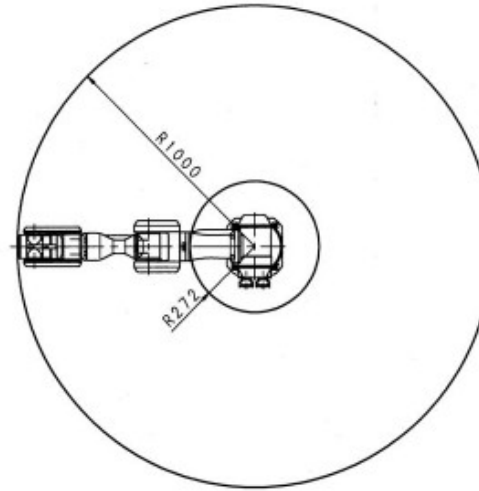
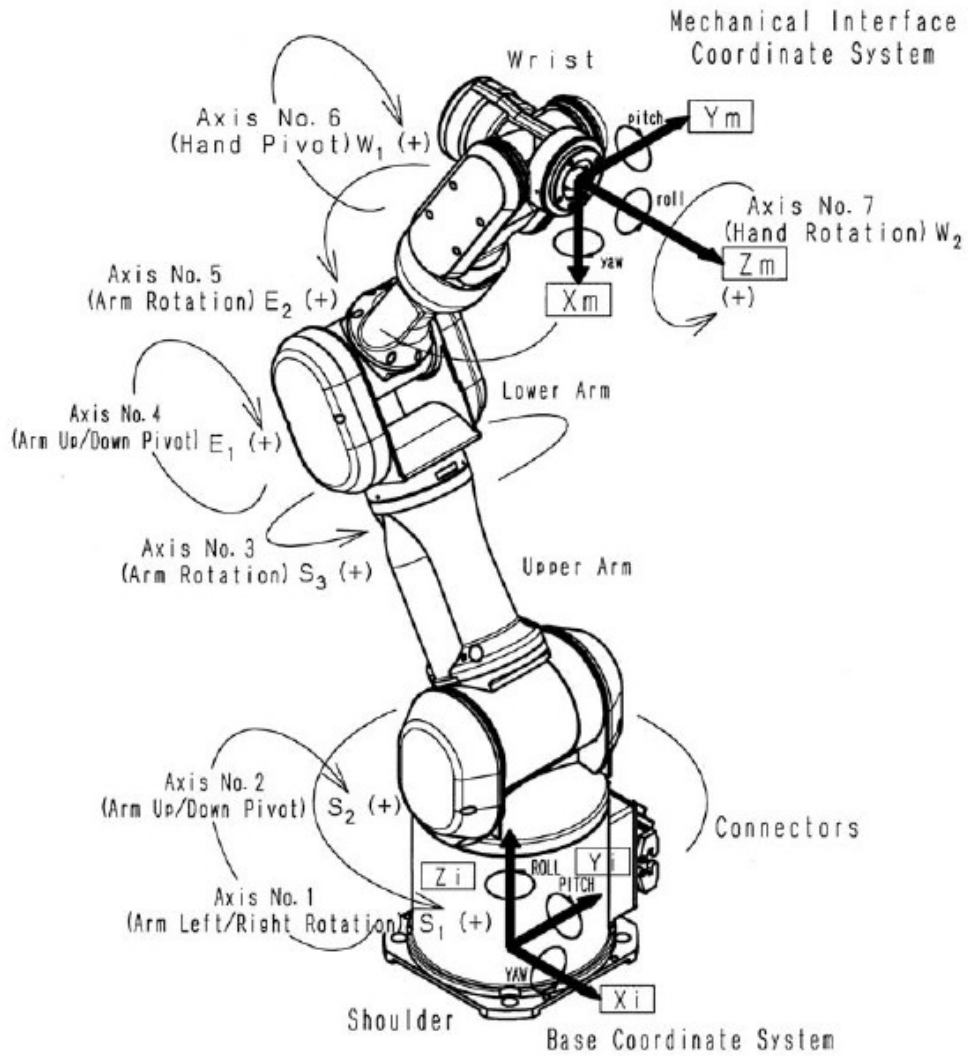
ERIGO
Hocoma
(CH)

Anklebot
MIT
(USA)



- Create a robotic platform and prototype for the rehabilitation of the human upper arm
- Implementation of complex control strategies to manage forces exchanged between the patient and the robot
- Virtual walls and surfaces to simulate real scenarios and virtual exercises
- High level user interface to interact with the machine and to create tridimensional immersive scenarios
- Integration of multisensorial input for the patient

- The main components of the experimental setup are:
 - Robot (Mitsubishi Pa10 – 7dof)
 - Possibility to by-pass its native control and to communicate directly with motor drivers
 - Force/Torque sensor (ATI Mini45)
 - To measure forces and torques exerted by the patient
 - Controller
 - PC-based control (GNU/Linux - Xenomai real-time kernel - C++)
 - GUI
 - High level GUI (Python)
 - 3D model
 - Virtual scene (VPython)

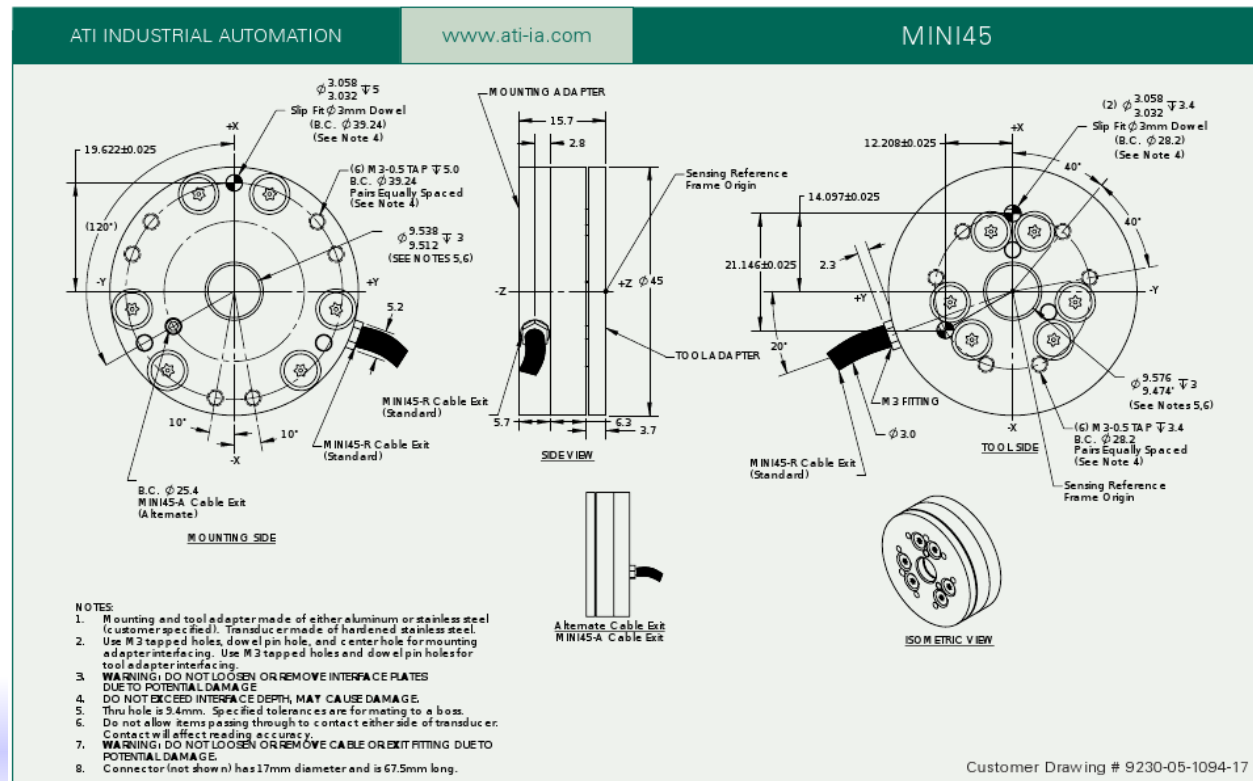


- Measures of forces and torques
- Right dimensioned for the robot and the selected application



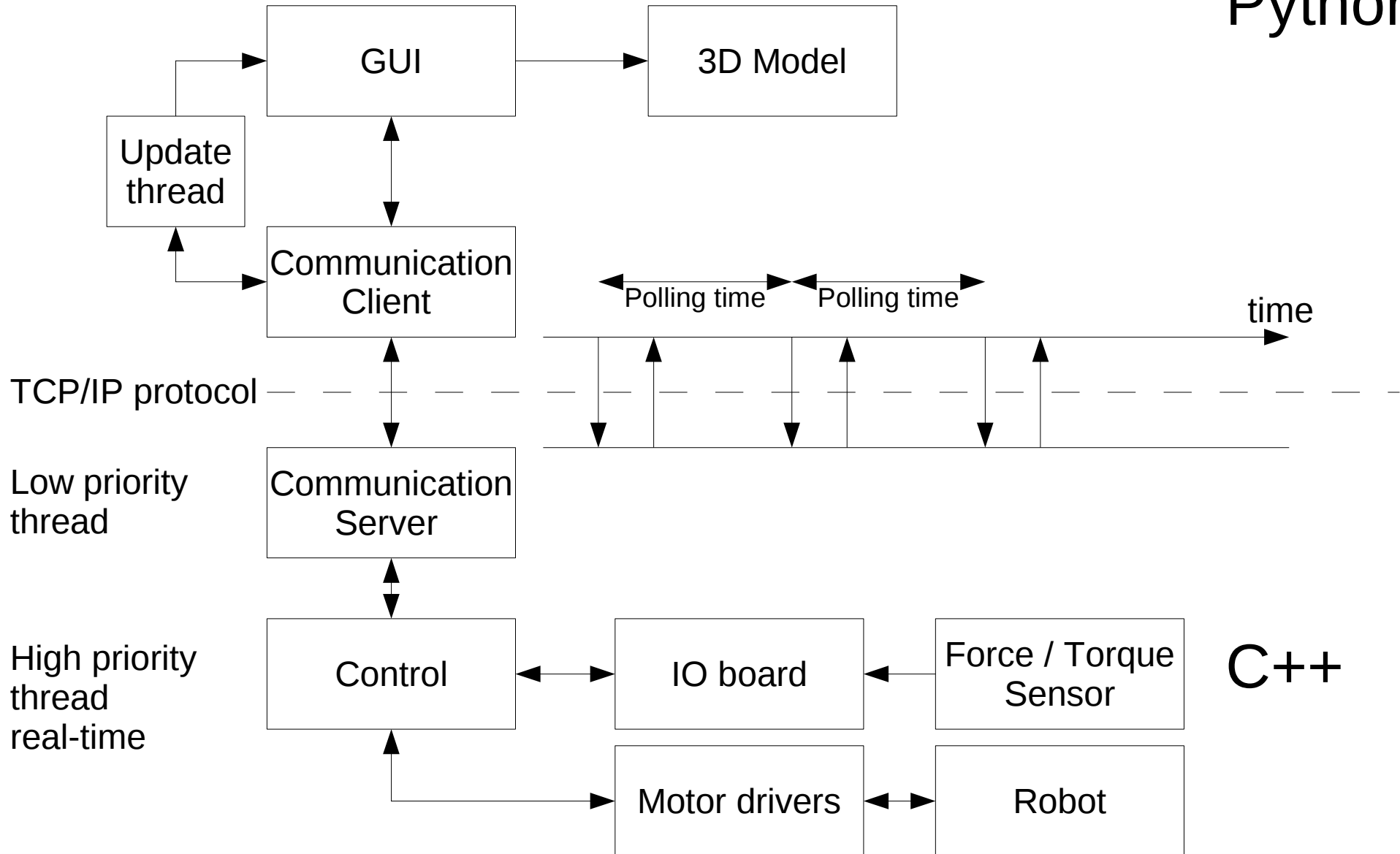
SENSING RANGES Axes	Calibrations	
	SI-145-5	SI-290-10
Fx, Fy (±N)	145	290
Fz (±N)	290	580
Tx, Ty (±Nm)	5	10
Tz (±Nm)	5	10

Single-Axis Overload	English	Metric
Fxy	±1100 lbf	±5100 N
Fz	±2300 lbf	±10000 N
Txy	±1000 lbf-in	±110 Nm
Tz	±1200 lbf-in	±140 Nm
Stiffness (Calculated)	English	Metric
X-axis & Y-axis force (Kx, Ky)	4.2x10 ⁵ lb/in	7.4x10 ⁷ N/m
Z-axis force (Kz)	5.6x10 ⁵ lb/in	9.8x10 ⁷ N/m
X-axis & Y-axis torque (Ktx, Kty)	1.5x10 ⁵ lbf-in/rad	1.7x10 ⁴ Nm/rad
Z-axis torque (Ktz)	3.1x10 ⁵ lbf-in/rad	3.5x10 ⁴ Nm/rad
Physical Specifications	English	Metric
Weight*	0.20 lb	92 g
Diameter (OD, ID)*	1.77 in, 0.373 in	45 mm, 9.5 mm
Height*	0.62 in	15.7 mm

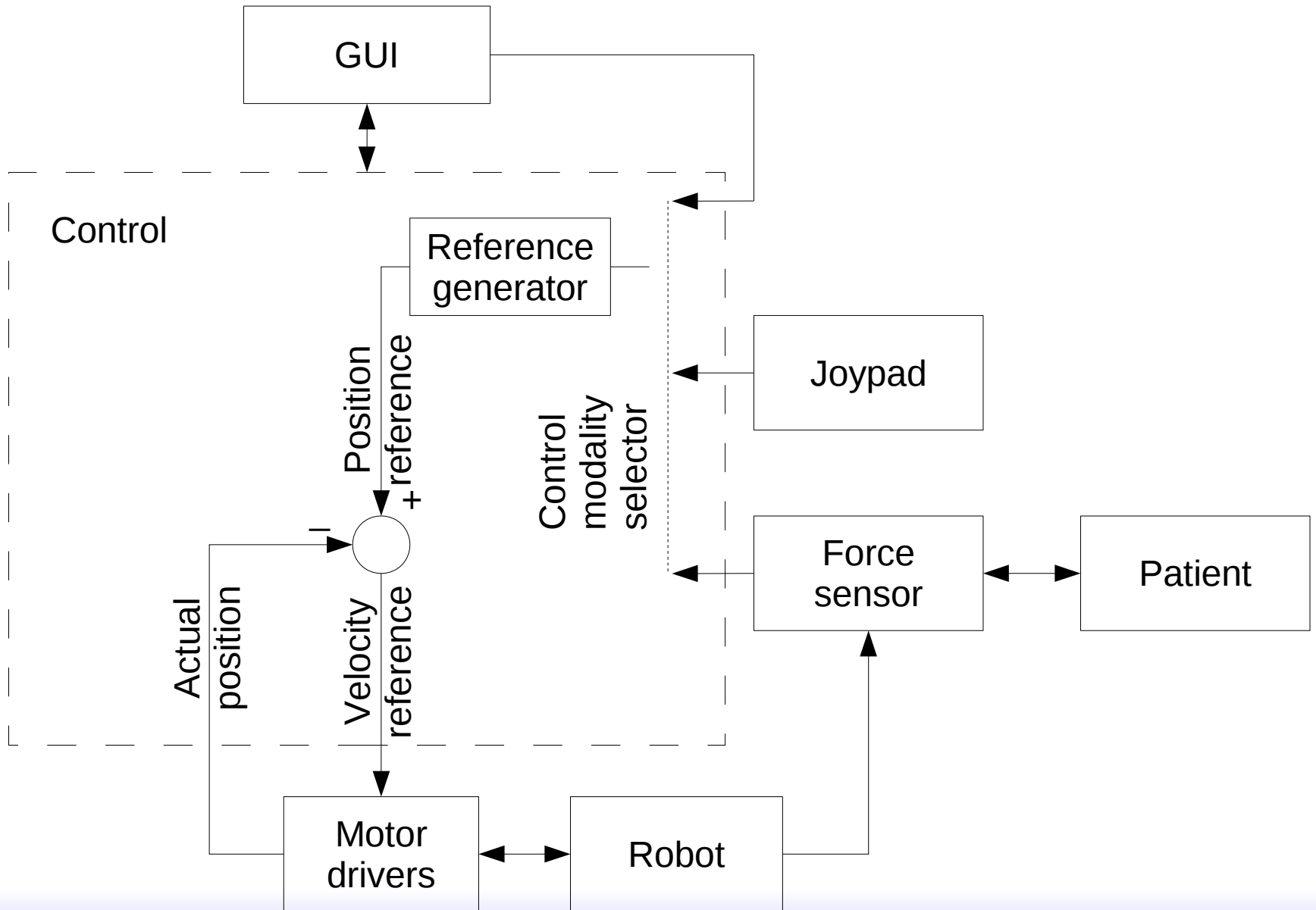


	Requirements	Implementation
CONTROL	Real-time high speed execution	GNU/Linux Debian (Xenomai kernel patch) - C++
GUI	Easy (re)configuration and high level functionalities. No real-time	Python with integration of several modules

Python



C++



- Functionalities:
 - Bi-directional communication with robot control
 - Robot command and programming
 - 3D virtual model
 - User-friendly
- Main components:
 - User interface: Qt4 python bindings
 - Communication: std library
 - Threads: QThread

- Robot command
 - Start, stop, reset, brakes release, robot parameters setting
- Select control modalities
 - GUI reference, force/torque sensor, ...
- Robot monitoring
 - Joint, TCP, Force, and patient's movements
- Control and application setting
 - Trajectories, springs, logs, paths, ...

GUI screenshot

The screenshot shows the Pa10 control interface with the following sections:

- Control:** Buttons for No Control, Jog Joint, Jog Tcp, Jog Tcp Curve, and Force Sensor.
- Joint:** A dropdown menu set to 0,00 and a 45% progress bar. Below are buttons for joints J1- through J7+.
- Tcp:** A dropdown menu set to 0,00 and buttons for X-, X+, Y-, Y+, Z-, Z+, Rho-, Rho+, Th-, Th+, and Psi-, Psi+.
- Springs:** A table with columns for m[kg], k[N/m], and c[Ns/m]. Rows include Tangent, Radial, and Circ, each with three input fields set to 0,00. Below are Set Parameters and Reload buttons.
- Trajectory:** A REL. BREAK button, a dropdown set to 0,00 with left and right arrow buttons, and Approach and Load Trajectory buttons.
- Bottom Panel:** A dropdown set to 0, Init Path, Spline, Line Point, Get Path, and ALARM RESET buttons.
- Right Panel:** A large vertical area containing START CONTROL, 3D Model, and ALARM RESET buttons.

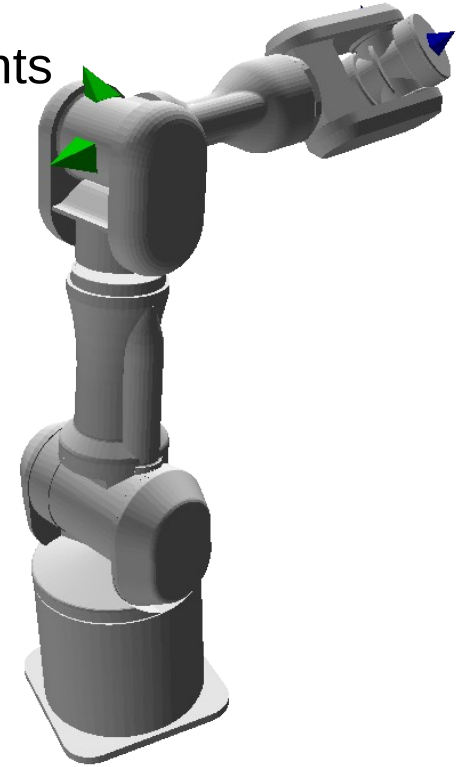
- Functionalities
 - Send command to the control
 - Receive data from the control
- Characteristics
 - Not real time and low frequency update cycle
 - Dedicated thread to manage continuous GUI update
- Implementation
 - Classes to manage outgoing and ingoing messages (data store and message creation and parsing)

```
class MessageToPa10():
    def SetCommand(self, cmd):
        # set local variable with data
    def ToSocket(self):
        # format local variable to be sentstrSocket = ""

class MessageFromPa10():
    def FromSocket(self, dataRcv):
        # parsing of message received

class ClientSocketThread (QtCore.QThread):
    def SendRcvMsg(self, msgToSend):
    def run(self):
        while not self.quit:
            self.msgRcvdNB = self.SendRcvMsg(self.msgToSendNB)
            self.emit(QtCore.SIGNAL("pySig"))
            sleep(0.1)
```

- Two main aims:
 - Debugging control code, simulating real robot movements
 - Immersive virtual scenario for patients
- Entities to be represented
 - Robot (especially for test and debug procedure)
 - Trajectories and surfaces to be followed
 - Auxiliary objects
- Implementation:
 - Vpython
 - Fast development of 3D (stereo) virtual models
 - STL import class
 - Import of complex and real shapes described in STL (stereolithography) CAD file format
 - VisualRobot class (based on Vpython ...)
 - Developed class for the representation and actuation of articulated mechanisms



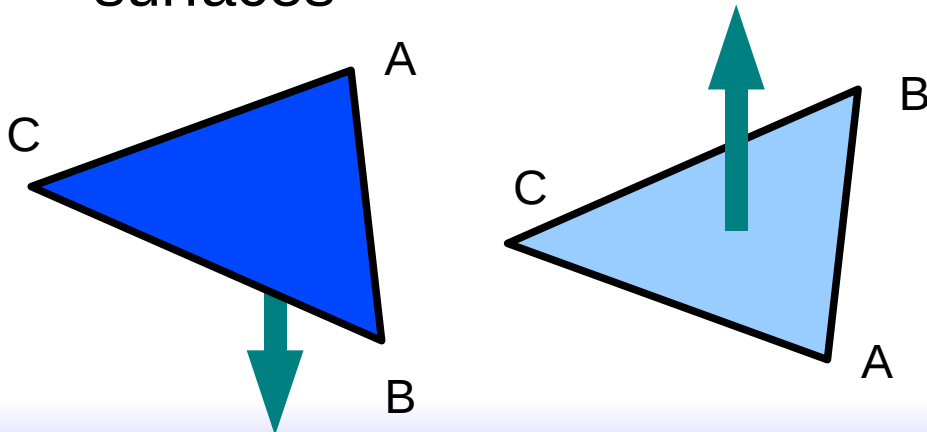
- An STL file consists of a list of facet data.
- Each facet is identified by a unit normal and by three vertices.
- Objects can be graphically represented through tessellation of their boundary surfaces

```

facet normal nx ny nz
  outer loop
    vertex Ax Ay Az
    vertex Bx By Bz
    vertex Cx Cy Cz
  endloop
endfacet
    
```

```

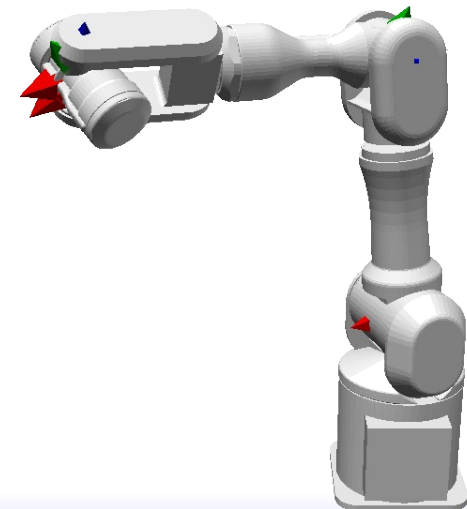
# example of code
class StlModel:
    def __init__(self, fileNames):
    def GetArrayData(self):
        return STLmodel
    
```



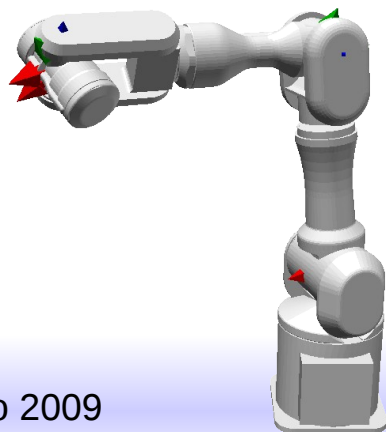
- Articulated mechanism
- Joint and actuator positions available from control
- Relative positions and movements of joints described by frames and their relative position/orientation
- Real objects represented by importing STL files

```

from StlModel import StlModel
from visual import *
class RobotModel():
    def __init__(self, _numAxis):
    def CreateTriad(self, targetFrame):
    def SetAxisMount(... ):
    def SetAxisAngle( ... ):
    def SetAxisPos( ... ):
    def AddStlGeometry(self, iAxis,
        fileNameList):
  
```



- Creation of composite objects
- All entities in the same frame moves rigidly
- Position and orientation of the frame can be modified through pos, x, y, z, axis, or up attributes
- Creation of articulated mechanisms



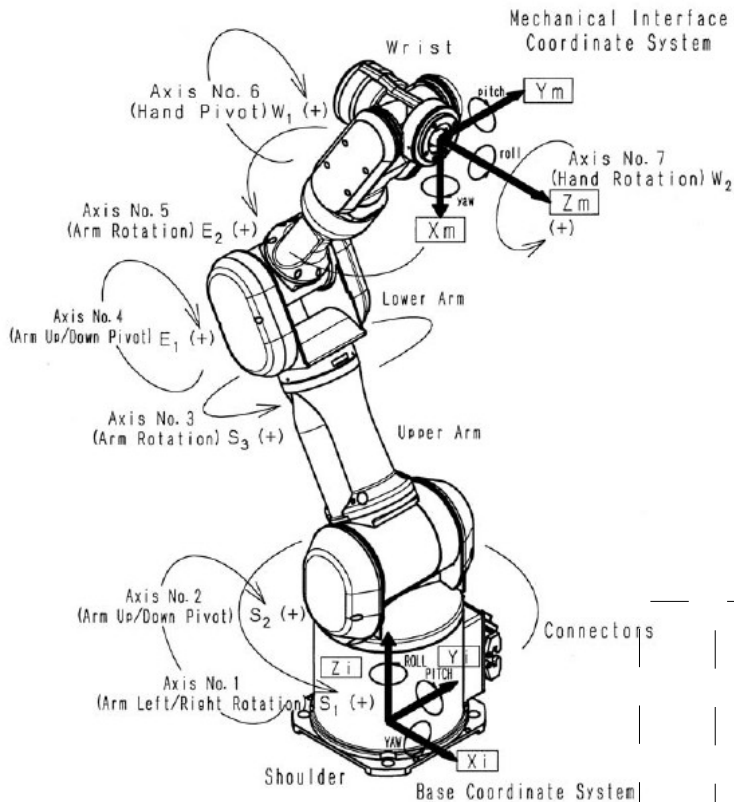
```

from visual import *

class RobotModel():
    def __init__(self, numAxis):
        # Frames definition
        self.links = [frame()]
        for iFrame in range(numAxis):
            self.links.append(frame())
            self.links[idFr+1].frame = self.links[idFr]

# position of STL geometry in kinematical frame
def SetAxMount(self, iAx, pos, orX, orY ):
    self.frameMountList[iAx].pos = pos
    self.frameMountList[iAx].axis = orX
    self.frameMountList[iAx].up = orY

# set position and angle of the kinematical frame
def SetAxisAngle(self, iFr, angle):
    c = cos(angle)
    s = sin(angle)
    self.links[iFr].axis=(c, s, 0)
    self.links[iFr].up=(-s, c, 0)
def SetAxisPos(self, iFr, pos):
    self.links[iFr].pos=pos
    
```



Structure of INI file

numLinks = nLinks

Pos = [(p1x, p1y, p1z), (p2x, p2y, p2z), ...]

OrX = [(C1xx, C1xy, C1xz), (C2xx, C2xy, C2xz), ...]

OrY = [(C1yx, C1yy, C1yz), (C2yx, C2yy, C2yz), ...]

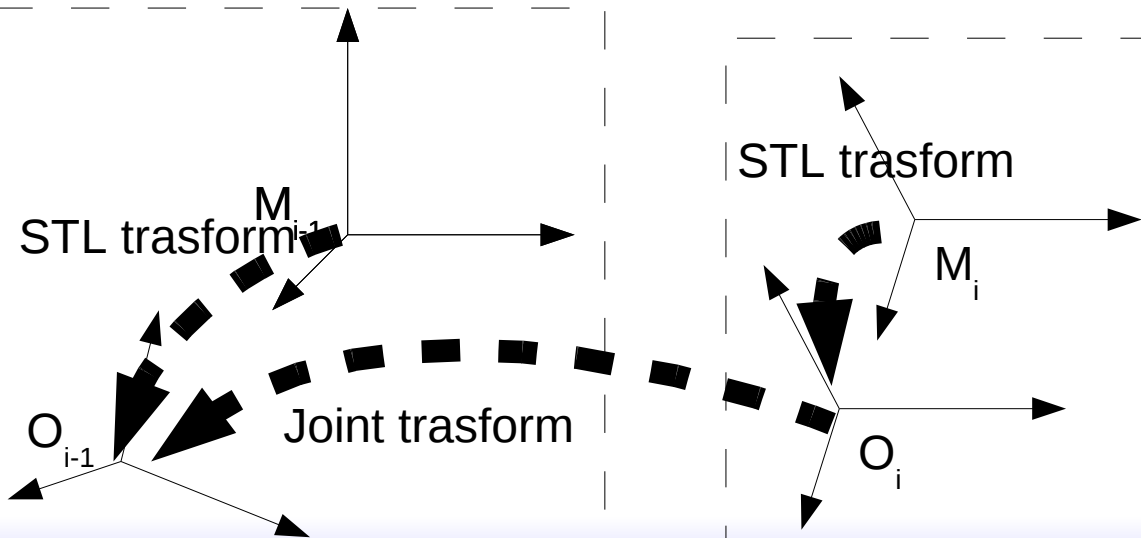
Sample of INI file (triad coincident with the previous one)

numLinks = 7

Pos = [(0, 0, 0), ...]

OrX = [(1, 0, 0), ...]

OrY = [(0, 1, 0), ...]



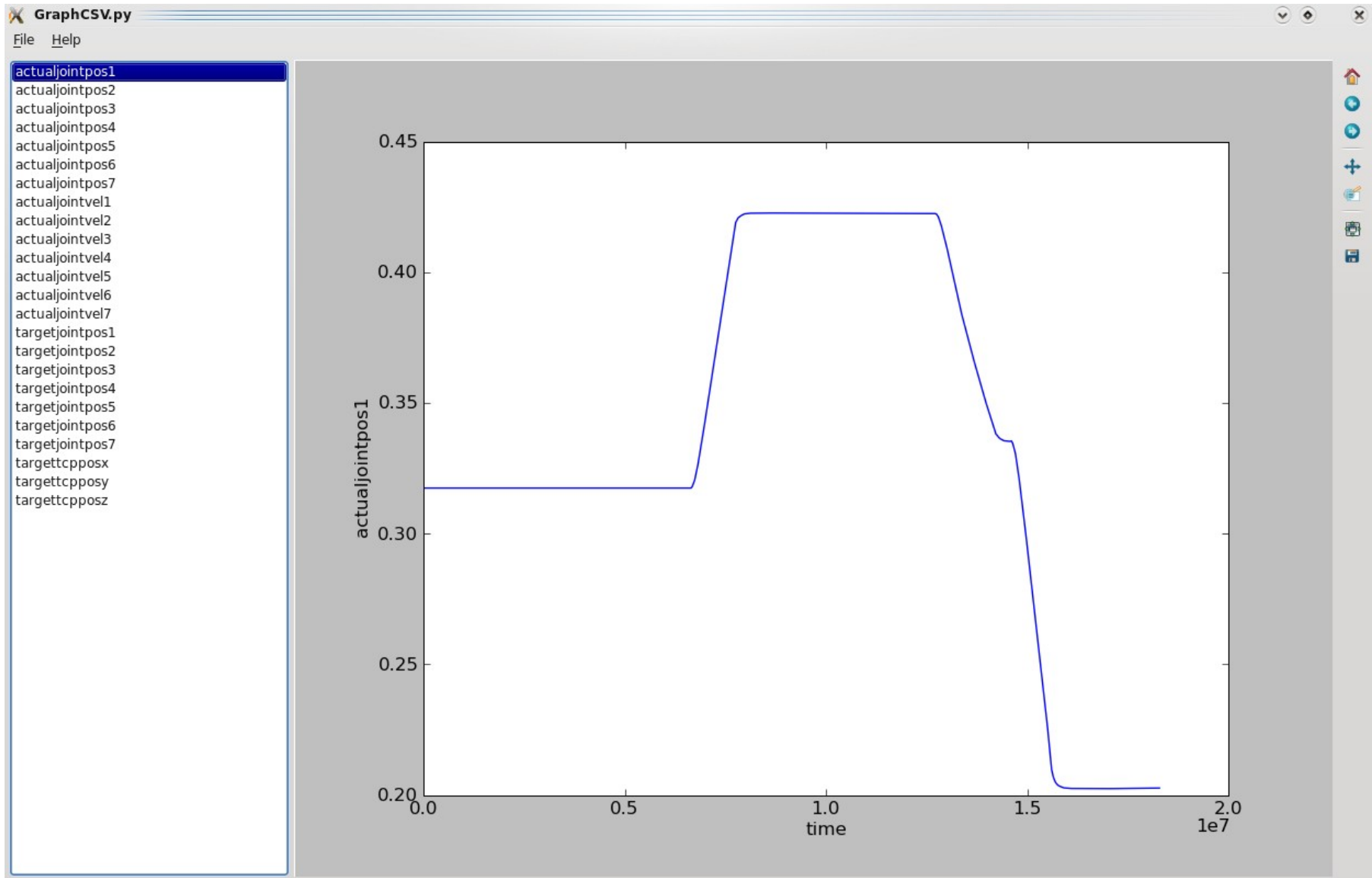
- Kinematical model of the robot
- Update of robot position
- Representation of additional entities, such as 3D path

```
from visual import *
from RobotModel import *

class VirtualModel:
    def __init__(self):
        # scene constructor
    def Update(self, listJPos):
        iJ=0
        for pos in listJPos:
            self.robotModel.SetAxisAngle(iJ, pos)
            iJ = iJ+1
    def DrawPathCurve(self, pointList):
        # functions for other entities
```

- Log files saved by the control
- State of the robot (e.g. actual/target joint position/velocity/torque, Ftsensor values)
- CSV file format with header row
- Integration with GUI
- matplotlib, pylab and backend for Qt4

Data plot (example)



Sample Video



- GUI for robot control, focused on rehabilitation application
- Highly customizable GUI and 3D virtual model
- Increase of usability of the GUI, adding functionalities dedicated to patient and therapist
- Modelling of virtual surfaces
- Integration of a redundant measuring system verify the actual position and increase the safety level
- Preliminary tests in a rehabilitation centre in few months

Thank you!

`matteo.malosio@itia.cnr.it`