PyRehabRob

Python Graphic User Interface for Robotic Rehabilitation

(Matteo Malosio – Nicola Pedrocchi – Lorenzo Molinari Tosatti)

PYCON ITALIA TRE

python™

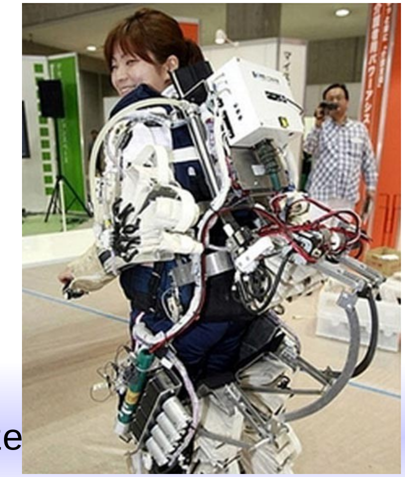
ITIA
Istituto di Tecnologie Industriali e Automazione
Consiglio Nazionale delle Ricerche

Outline

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

Robots

- 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



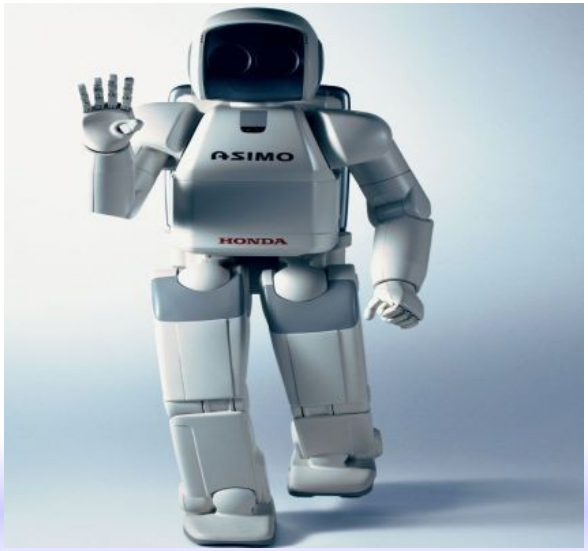
Robot kinematics

- Various possible kinematical structures
  - Serial

    large workspace, low stiffness and dynamics
  - Parallel

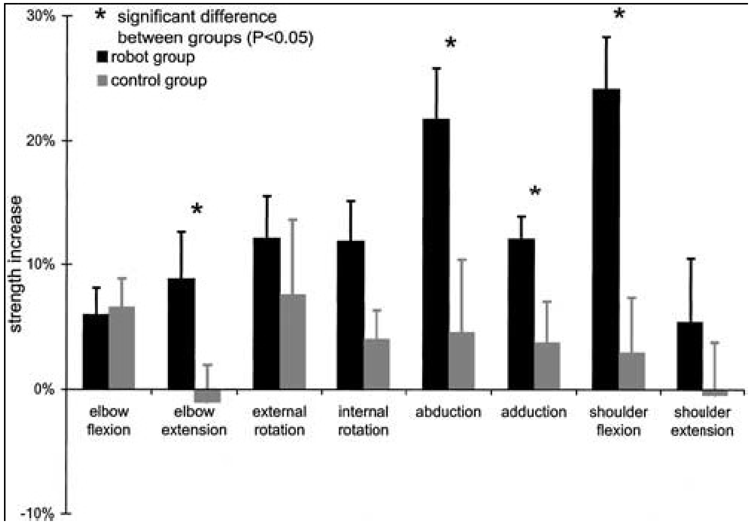
    small workspace, high stiffness and dynamics
  - Humanoid

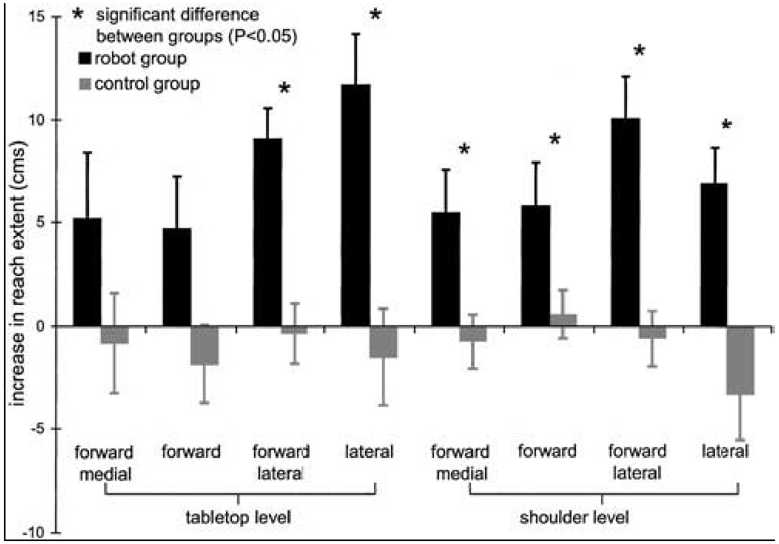
    service, games, social impact, mobiles



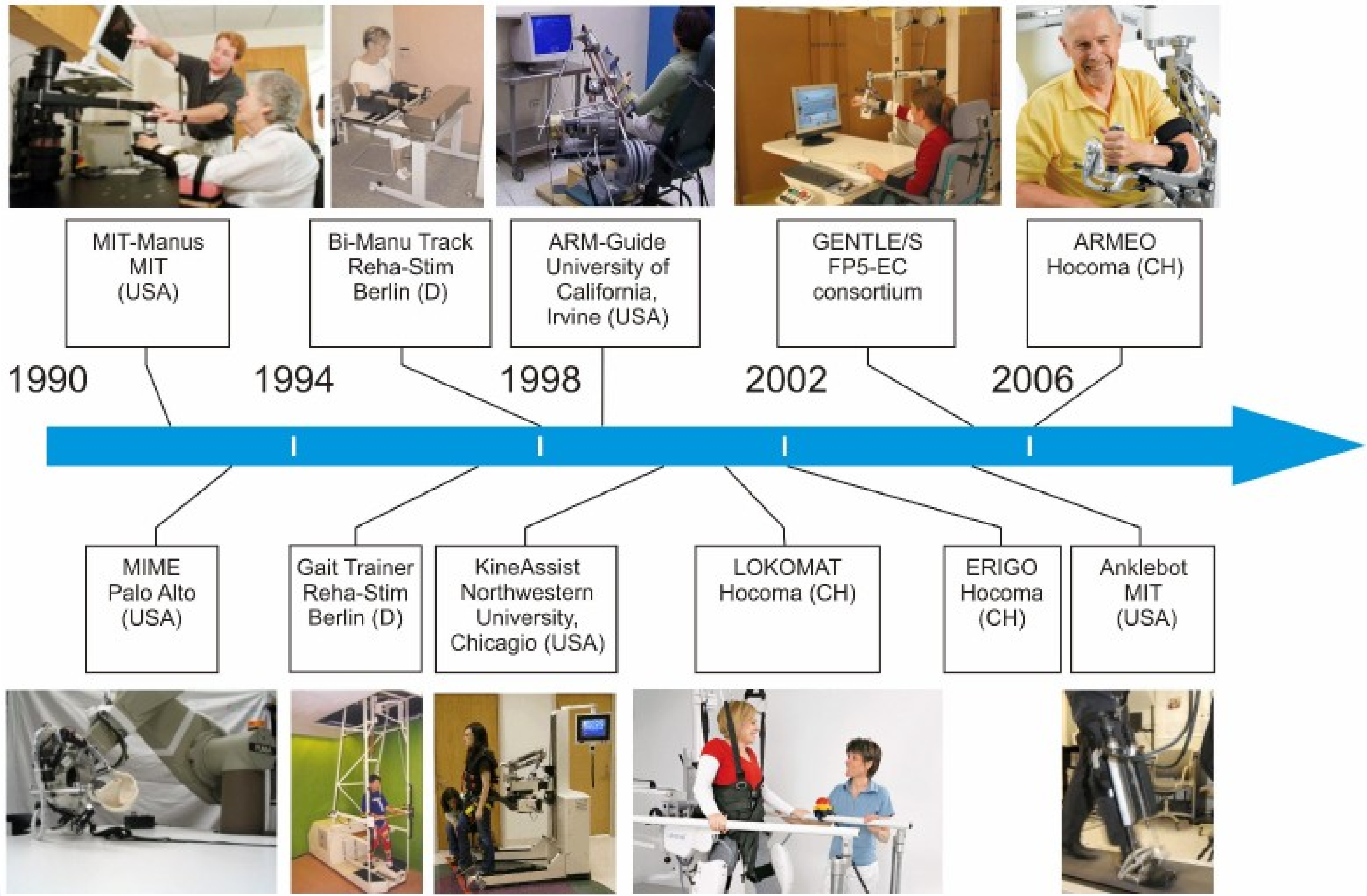
Rehabilitation robotics

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





Robotized systems for rehabilitation



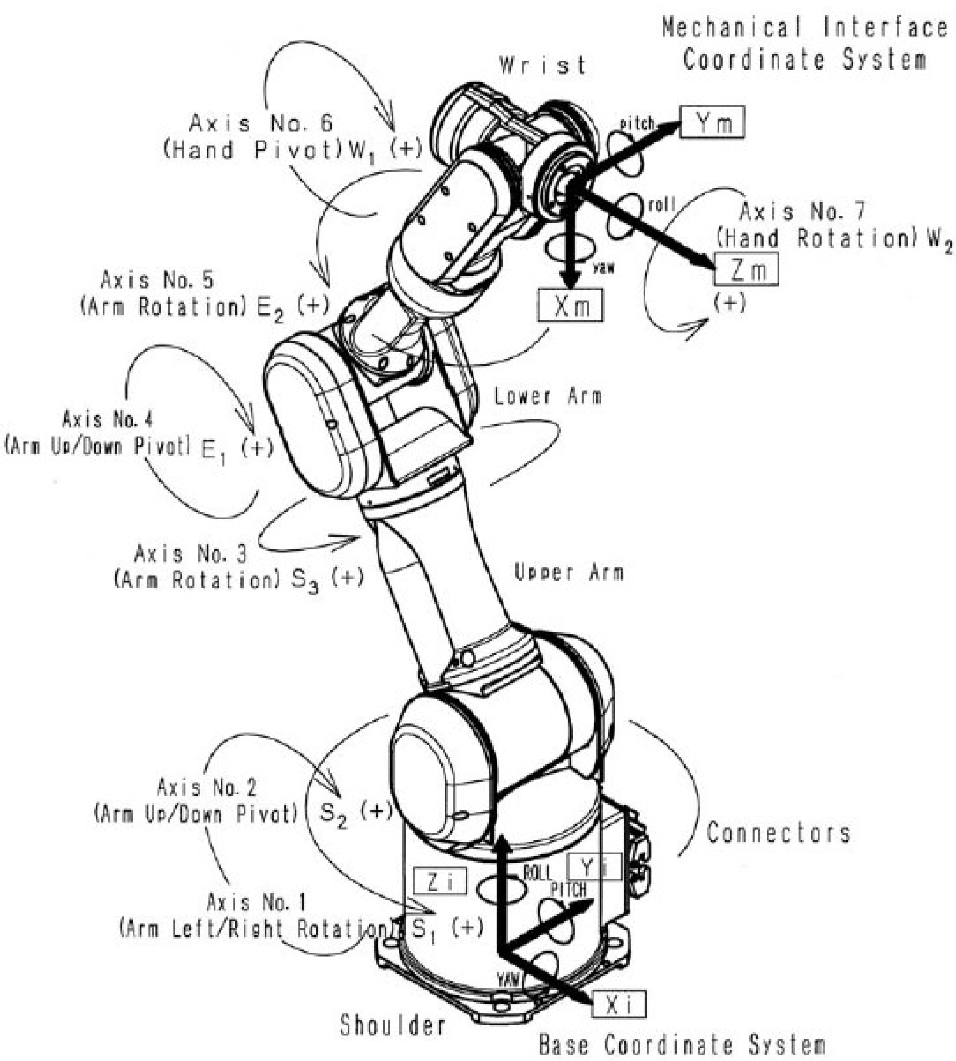
Aims

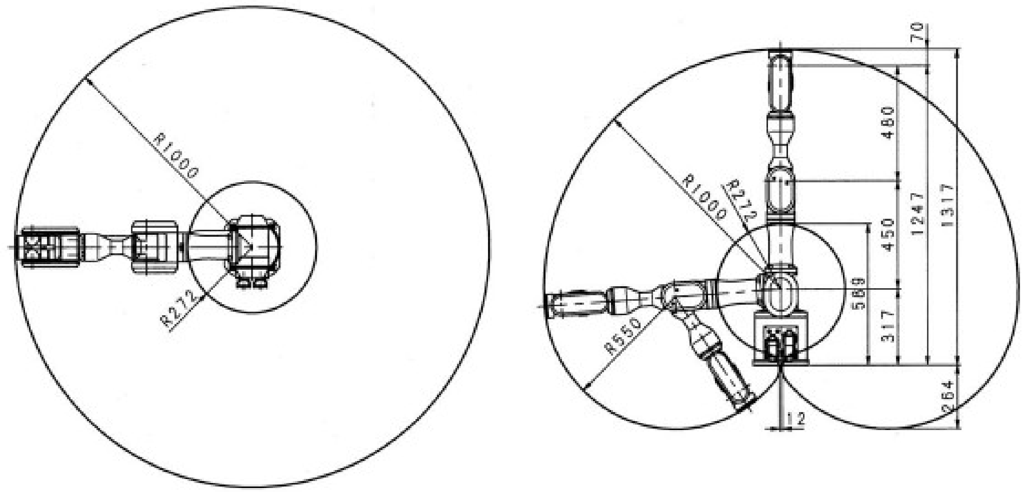
- Create a robotic platform and prototype for the rehabilitation of the human upper arm
- Implementation of complex control strategies to manage forces exchanged between the patient and the robot
- Virtual walls and surfaces to simulate real scenarios and virtual exercises
- High level user interface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

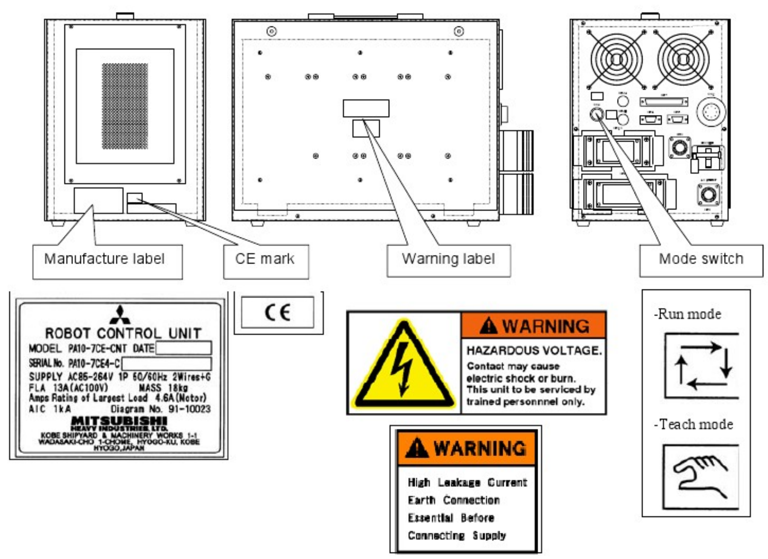
Experimental setup

- 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)

Robot


Mechanical Interface Coordinate System
Wrist
Axis No. 6 (Hand Pivot) W_1 (+)
pitch
Ym
roll
Axis No. 7 (Hand Rotation) W_2
Zm
(+)
yaw
Xm
Axis No. 5 (Arm Rotation) E_2 (+)
Lower Arm
Axis No. 4 (Arm Up/Down Pivot) E_1 (+)
Axis No. 3 (Arm Rotation) S_3 (+)
Upper Arm
Axis No. 2 (Arm Up/Down Pivot) S_2 (+)
Connectors
Zi
ROLL
Yi
PITCH
Axis No. 1 (Arm Left/Right Rotation) S_1 (+)
YAW
Xi
Shoulder
Base Coordinate System


R1000
R272
R1000
R272
R550
70
480
450
589
317
1247
1317
264
12


Manufacture label
CE mark
Warning label
Mode switch
ROBOT CONTROL UNIT
MITSUBISHI
WARNING
HAZARDOUS VOLTAGE.
Contact may cause electric shock or burn.
This unit to be serviced by trained personnel only.
WARNING
High Leakage Current
Earth Connection
Essential Before
Connecting Supply
-Run mode
-Teach mode

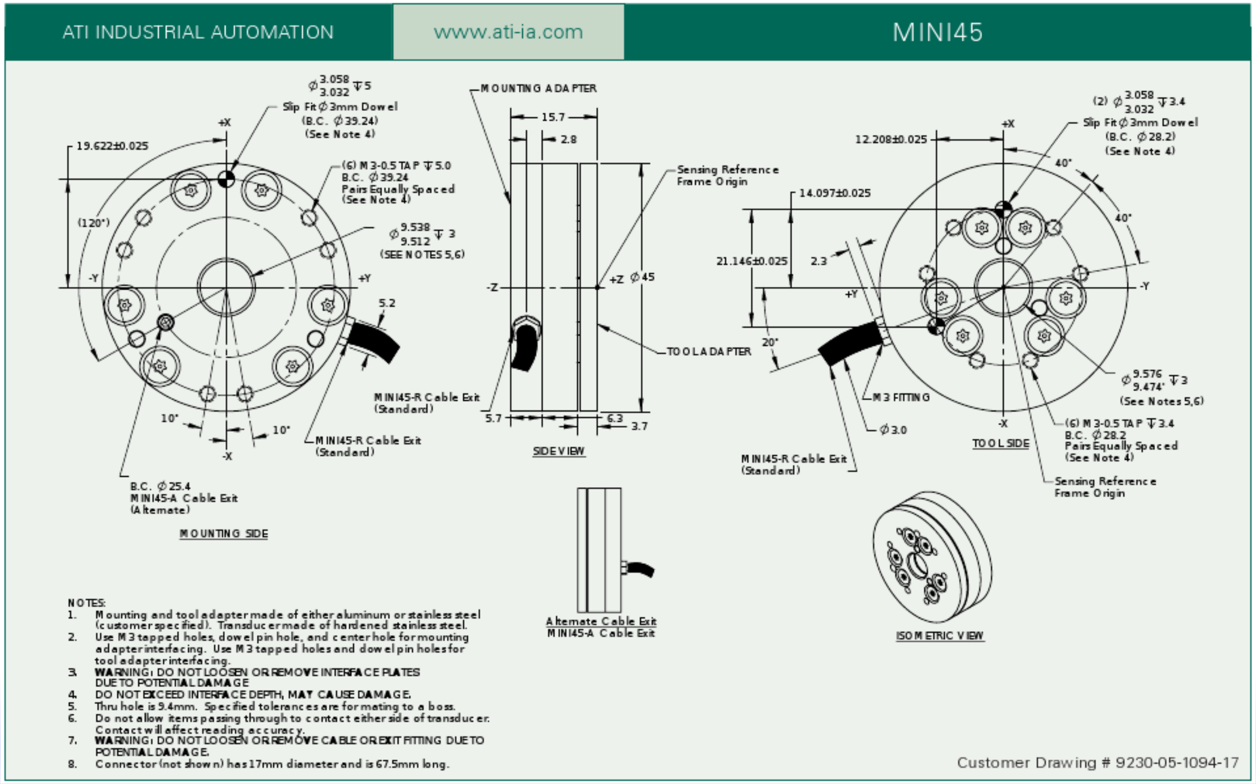
Force/Torque sensor

ITIA Istituto di Tecnologie Industriali e Automazione
Consiglio Nazionale delle Ricerche

- 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

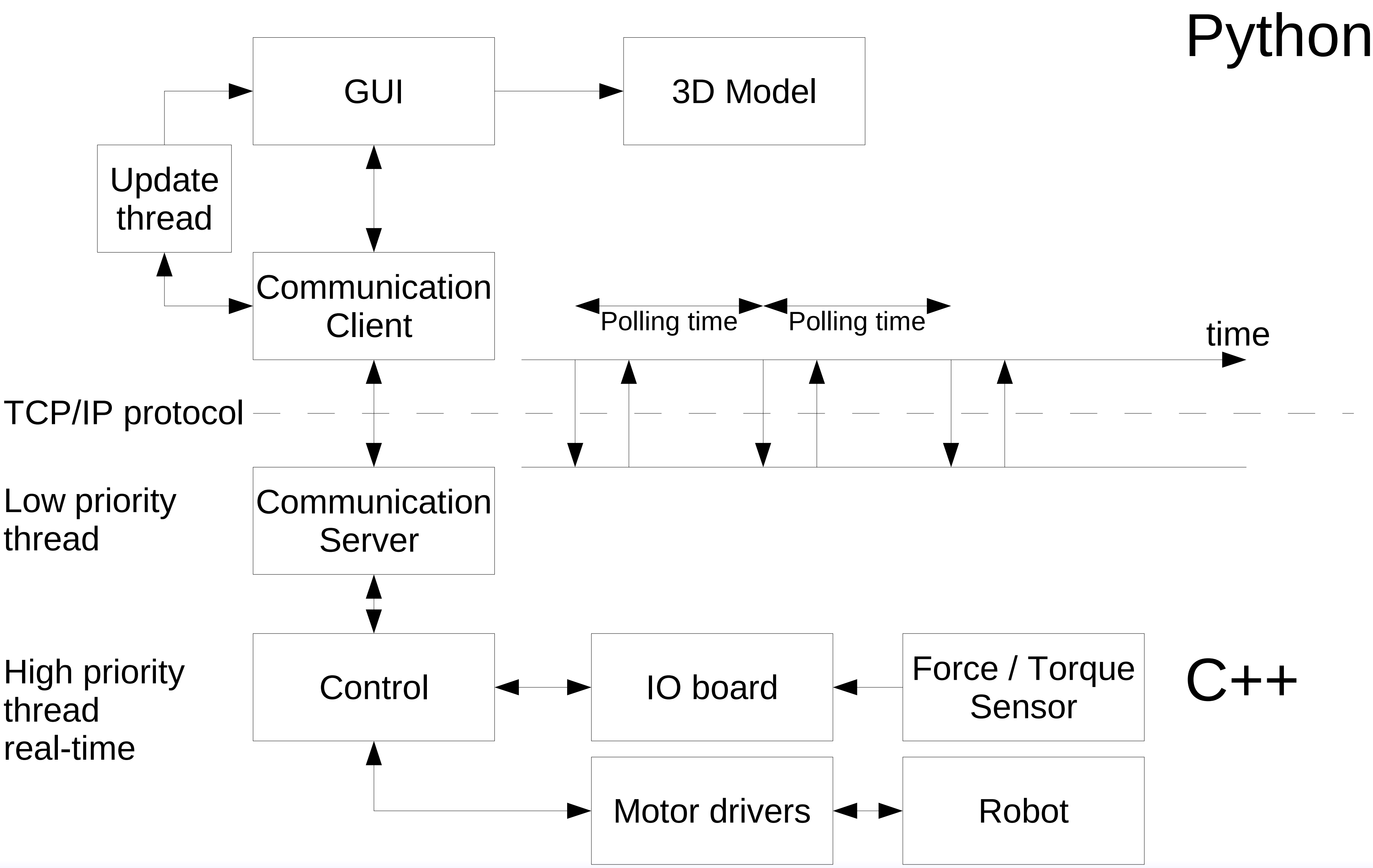
Single-Axis Overload	English	Metric
Fxy	±1100 lbf	±5100 N
Fz	±2300 lbf	±10000 N
Txy	±1000 lbf-in	±110 Nm
Tz	±1200 lbf-in	±140 Nm
Stiffness (Calculated)	**English**	**Metric**
X-axis & Y-axis force (Kx, Ky)	4.2×10^5 lb/in	7.4×10^7 N/m
Z-axis force (Kz)	5.6×10^5 lb/in	9.8×10^7 N/m
X-axis & Y-axis torque (Ktx, Kty)	1.5×10^5 lbf-in/rad	1.7×10^4 Nm/rad
Z-axis torque (Ktz)	3.1×10^5 lbf-in/rad	3.5×10^4 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



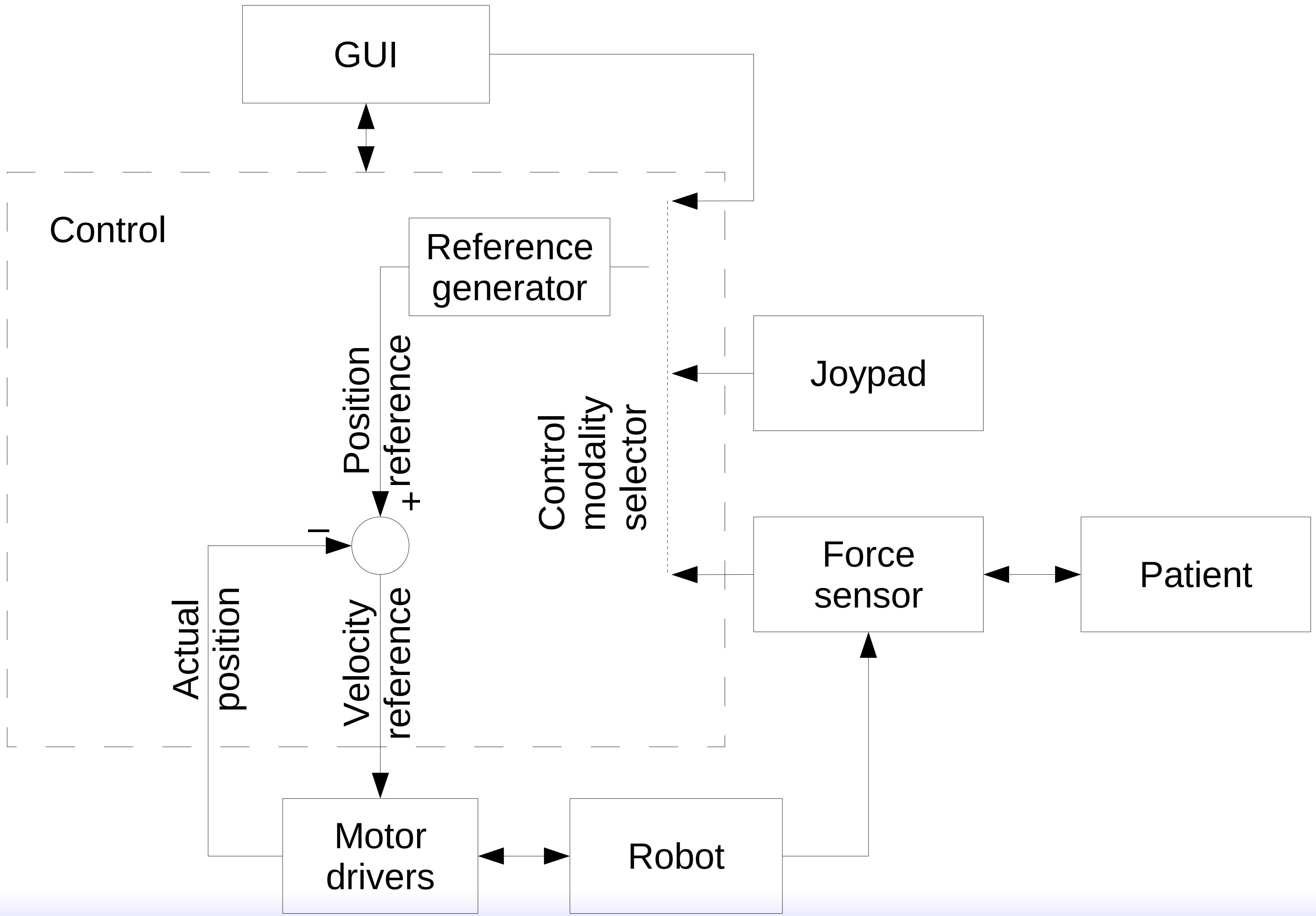
Software characteristics

	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

Control structure scheme



Low level control structure



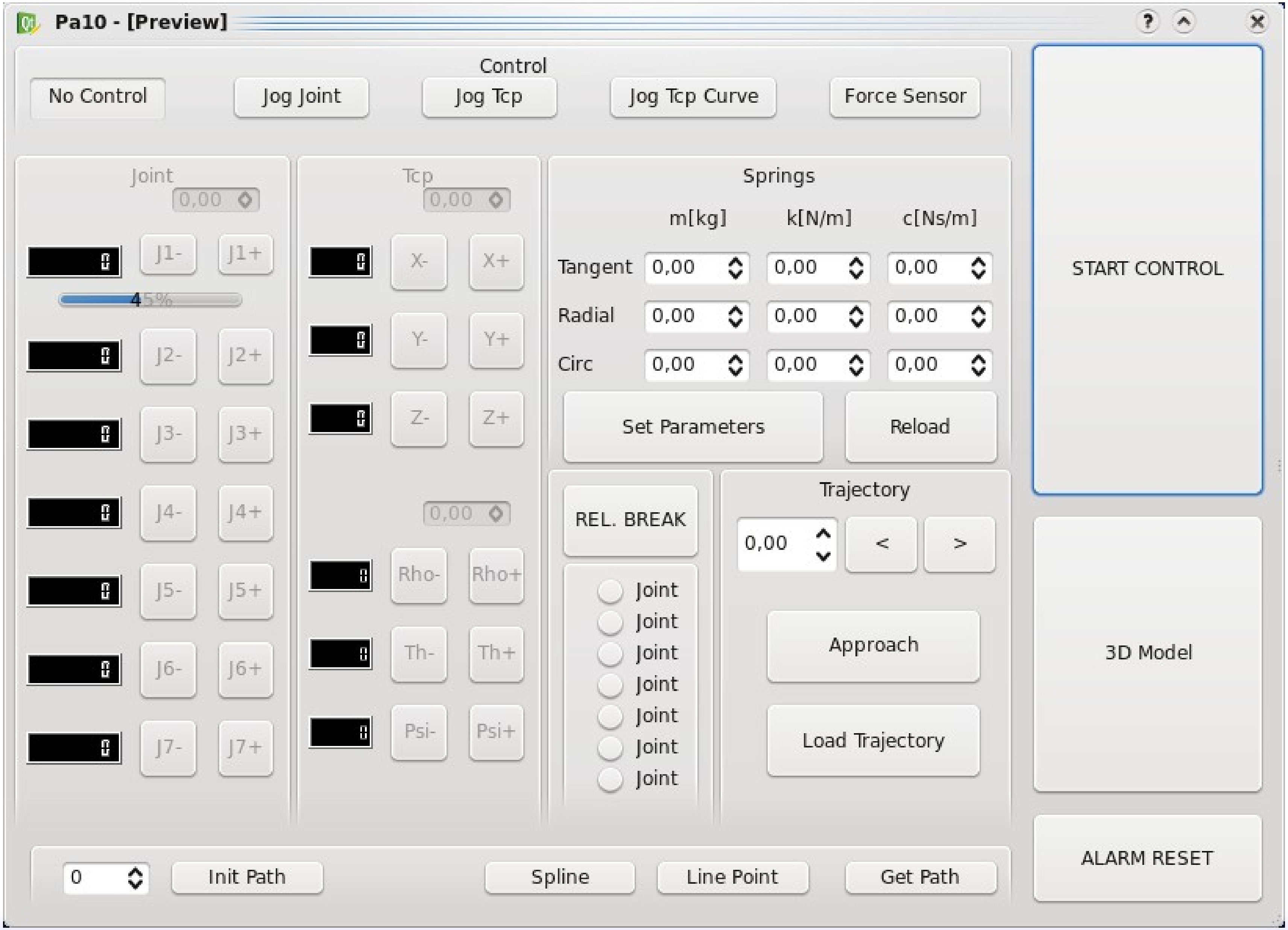
GUI

- 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

User interface

- 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



TCP/IP Communication

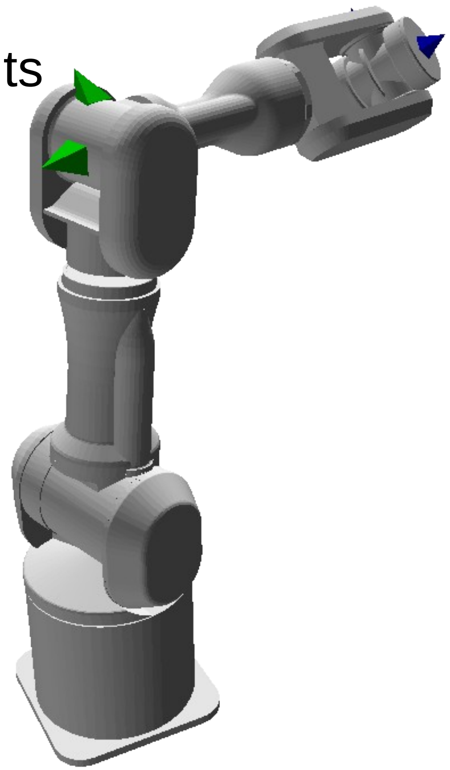
- Functionalities
  - Send command to the control
  - Receive data from the control
- Characteristics
  - Not real time and low frequency update cycle
  - Dedicated thread to manage continuous GUI update
- Implementation
  - Classes to manage outgoing and ingoing messages (data store and message creation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

TCP/IP Communication

```
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)
```

3D model

- 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 (stereolitography) CAD file format
  - VisualRobot class (based on Vpython ...)
    - Developed class for the representation and actuation of articulated mechanisms

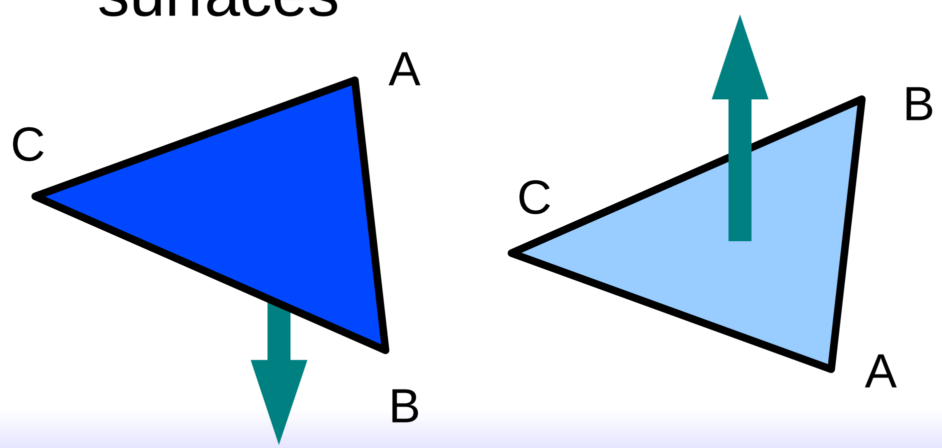


STL format

- 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
```

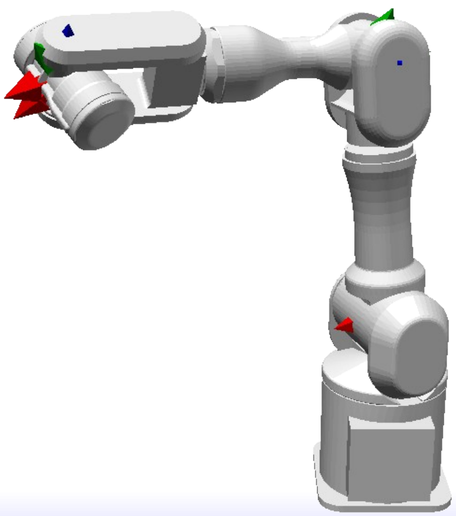
```
# example of code
class StlModel:
    def __init__(self, fileNames):
    def GetArrayData(self):
        return STLmodel
```



Robot model

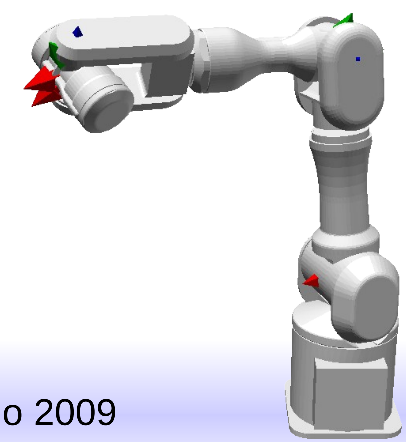
- Articulated mechanism
- Joint and actuator positions available from control
- Relative positions and movements of joints described by frames and their relative position/orientation
- Real objects represented by importing STL files

```
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):
```



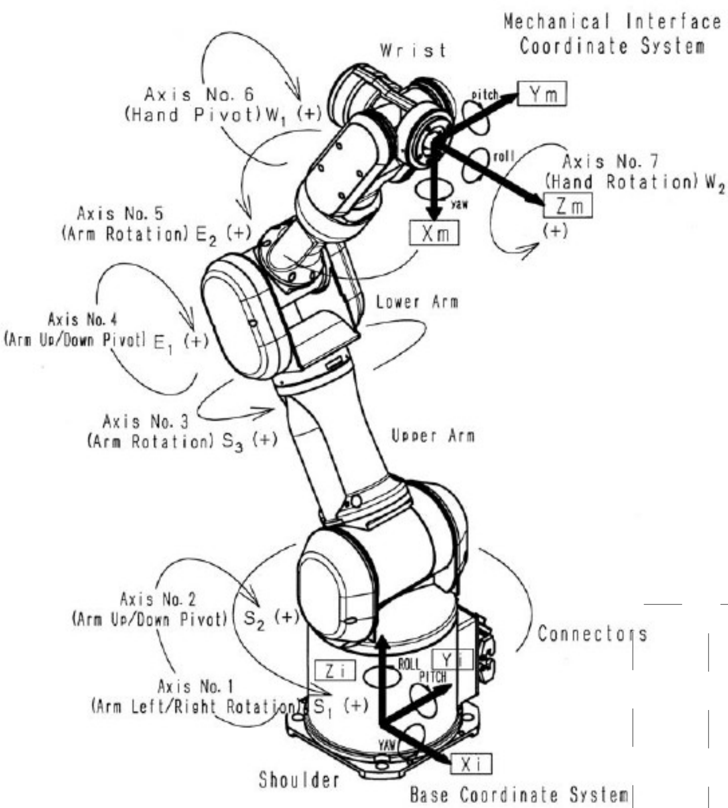
Robot links and joints

- 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

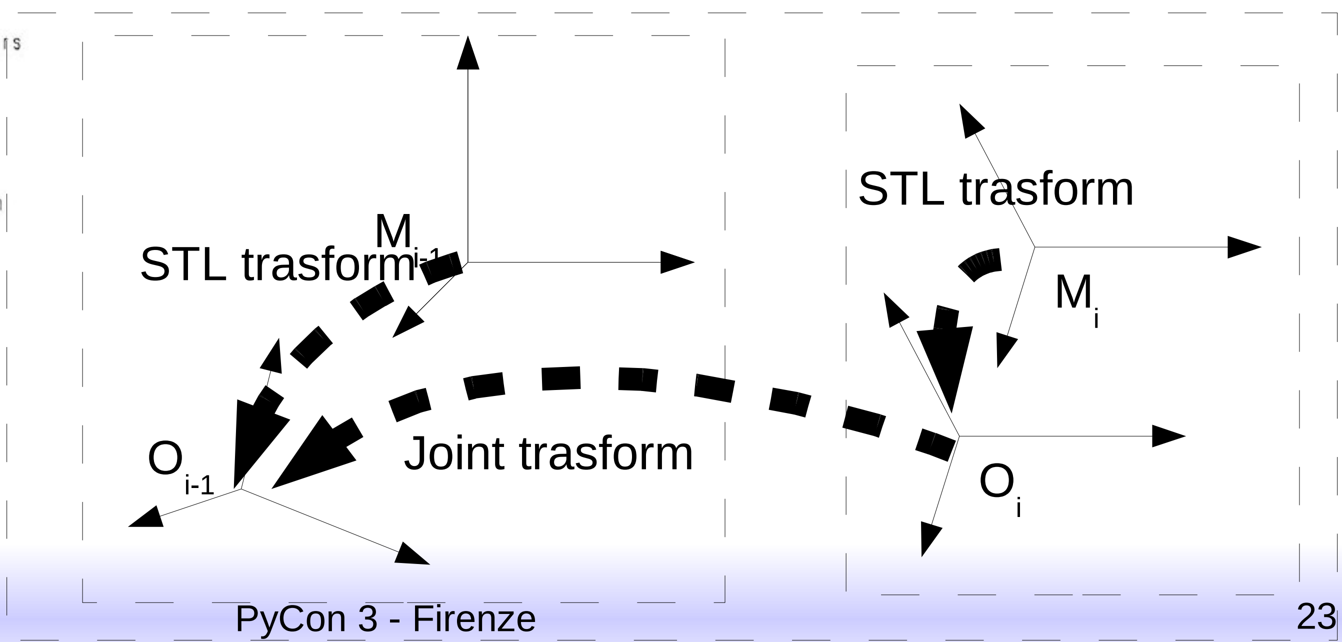


```
from visual import *

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
```

Robot links and joints




```
# Structure of INI file
numLinks = nLinks
Pos = [(p1x,p1y,p1z),(p2x,p2y,p2z),...]
OrX = [(c1xx,c1xy,c1xz),(c2xx,c2xy,c2xz),...]
OrX = [(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),...]
OrX = [(0,1,0),...]
```



Virtual model

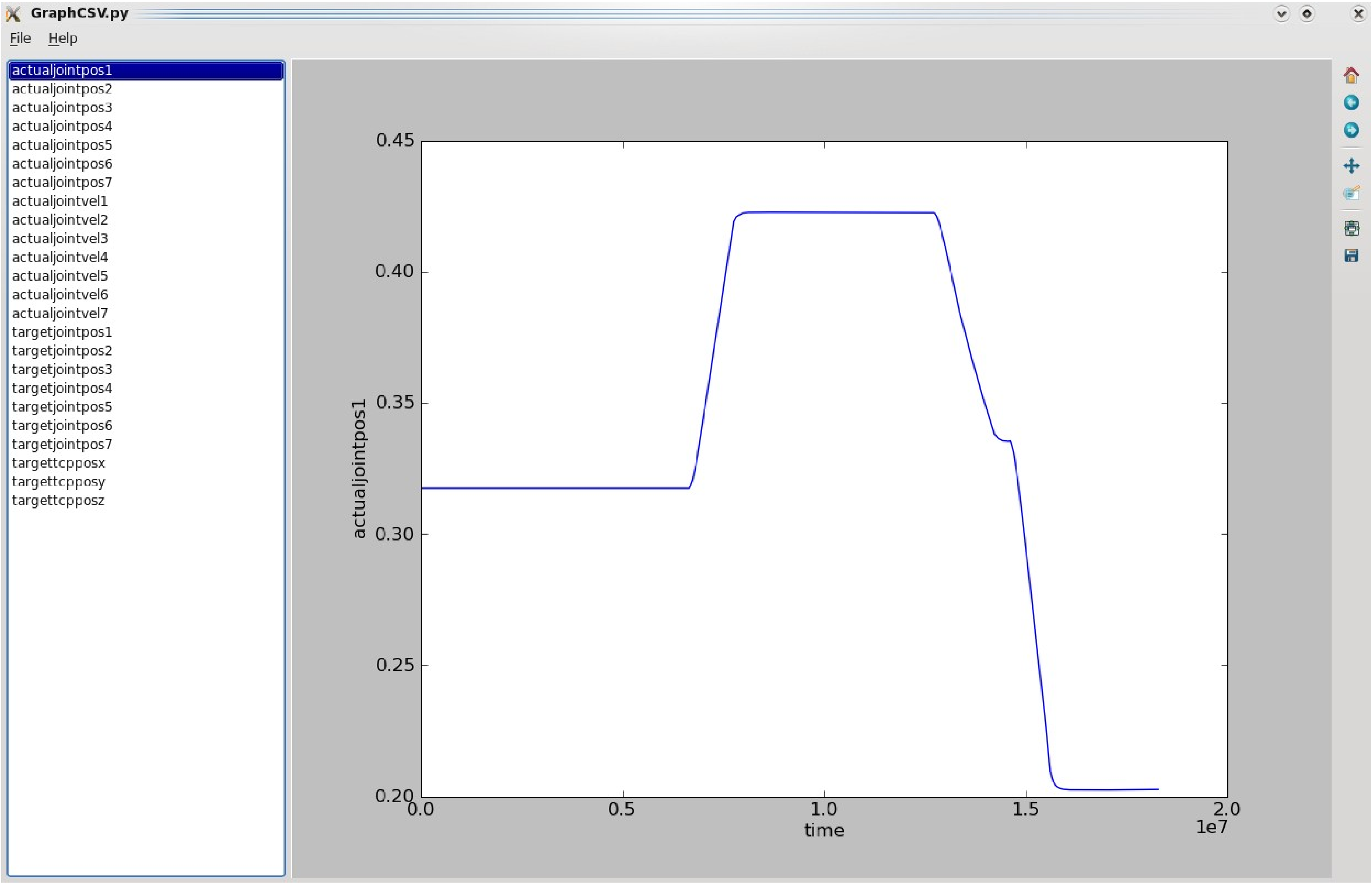
- Kinematical model of the robot
- Update of robot position
- Representation of additional entities, such as 3D path

```
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
```

Data plot

- 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

Conclusions and future works

- 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