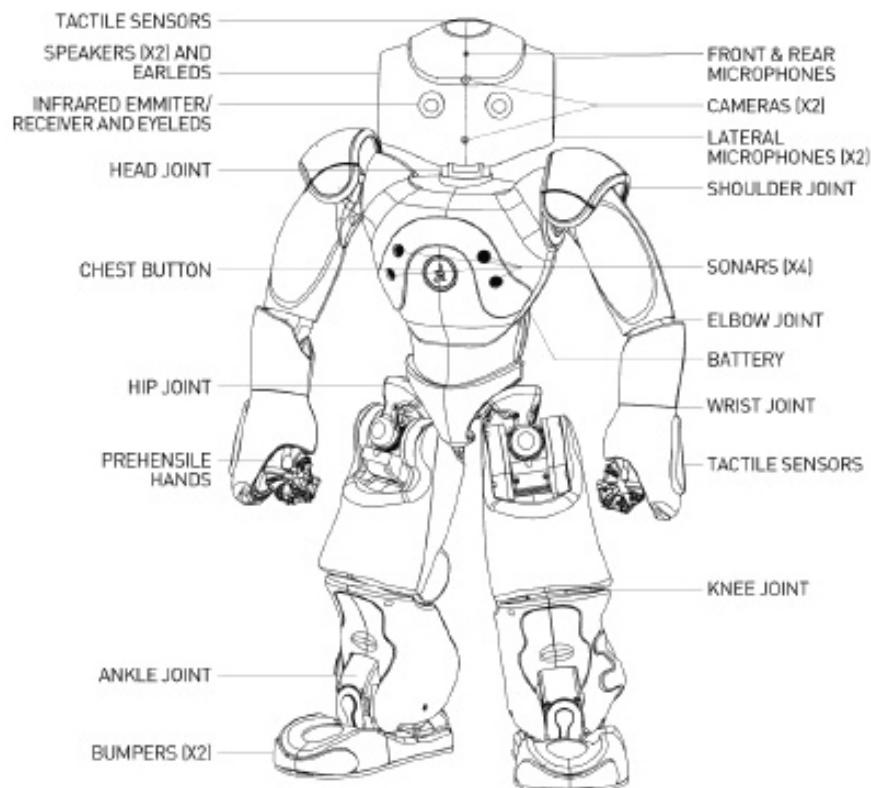


NAO

Programmare un robot umanoide

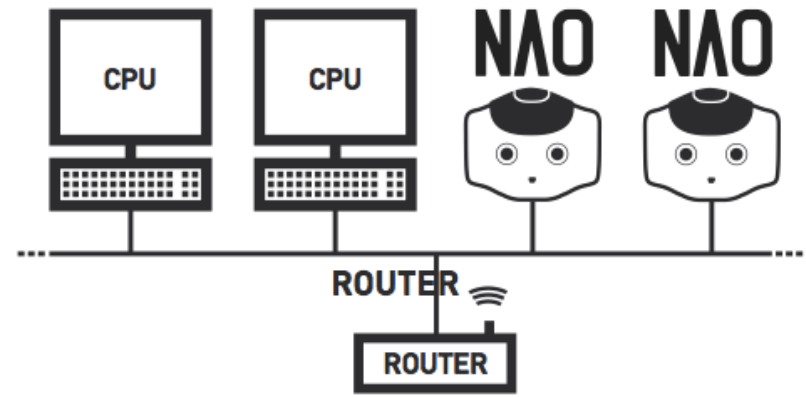
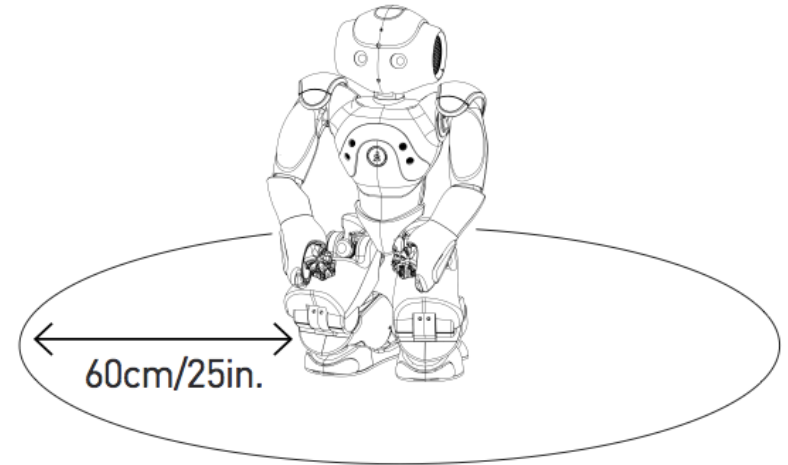


Nao: come è fatto?

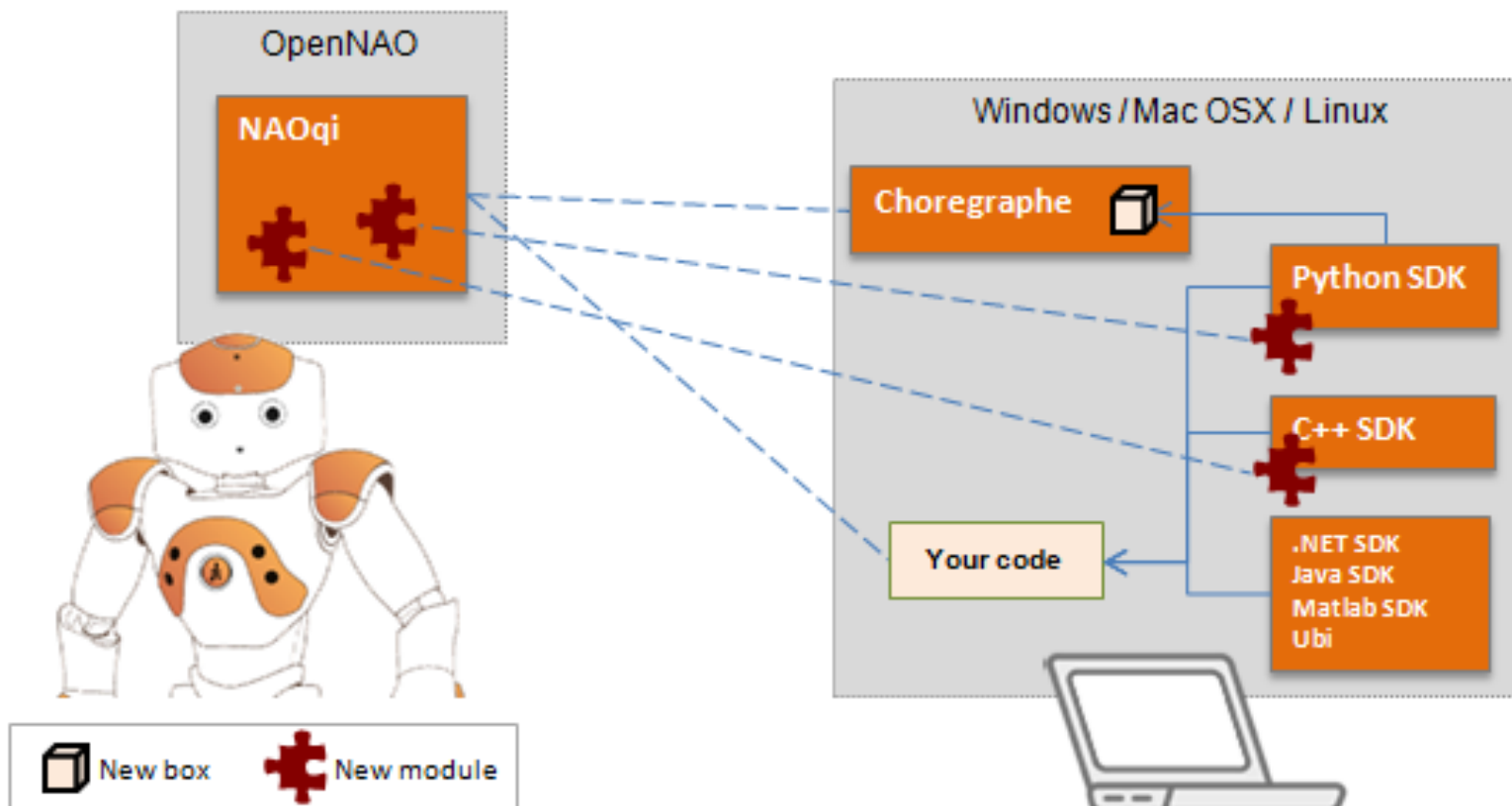


Posizionamento e connessione

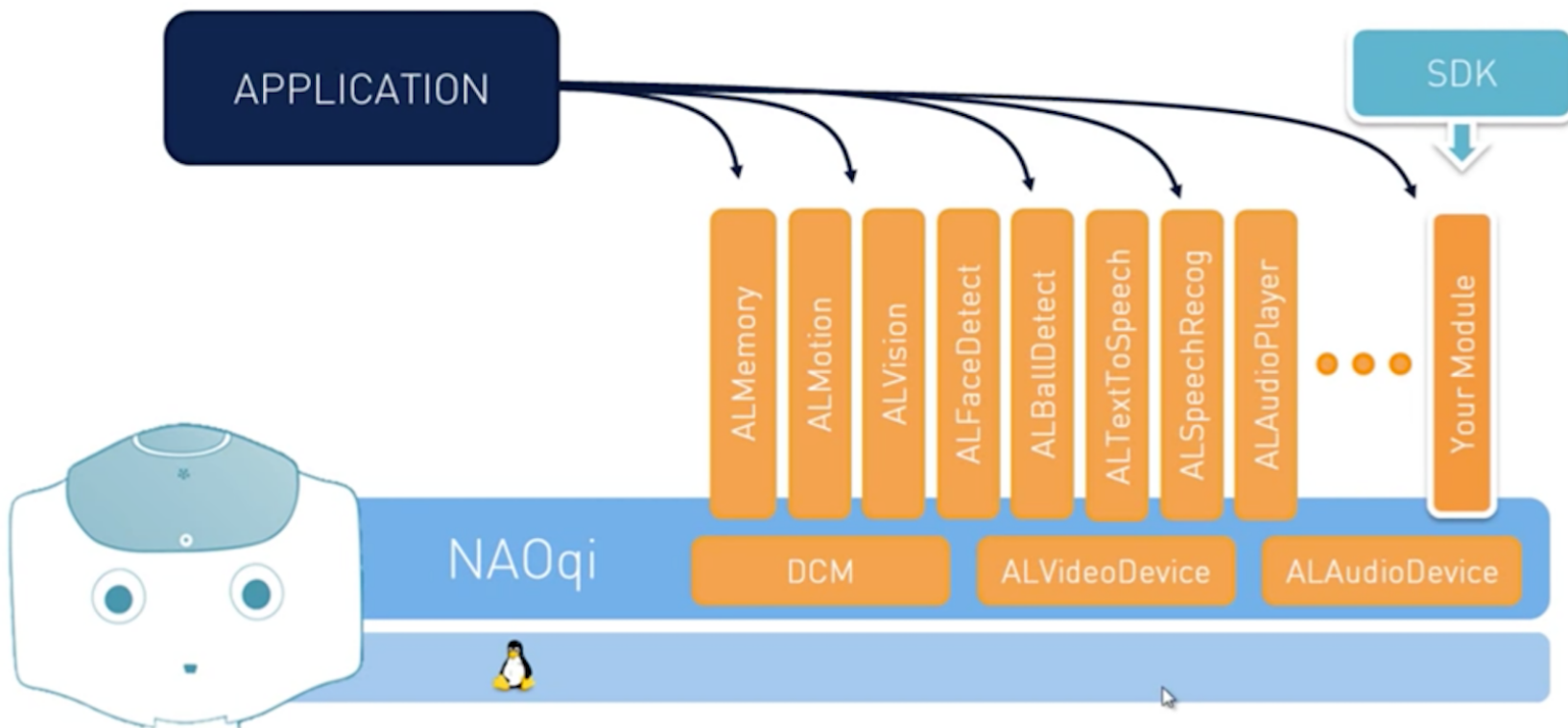
- Assicurarsi che ci sia sufficiente spazio intorno al robot. Posizionarlo preferibilmente sul pavimento.
- Assicurarsi che NAO sia connesso alla rete (ethernet o WiFi)



NAO: come funziona?



Perchè programmare in Python



Controllo da remoto



Say("Hello everyone! ...")

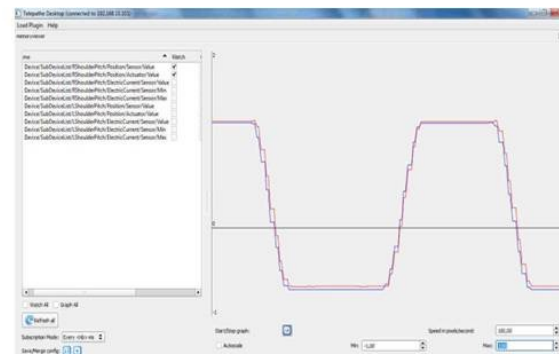
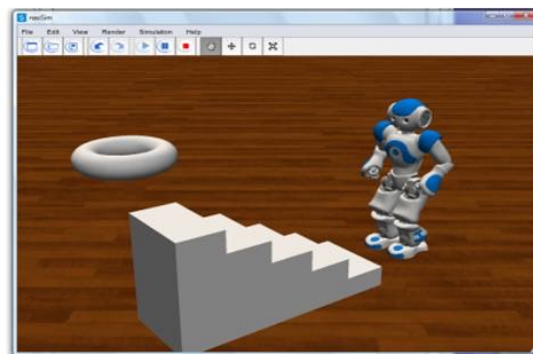
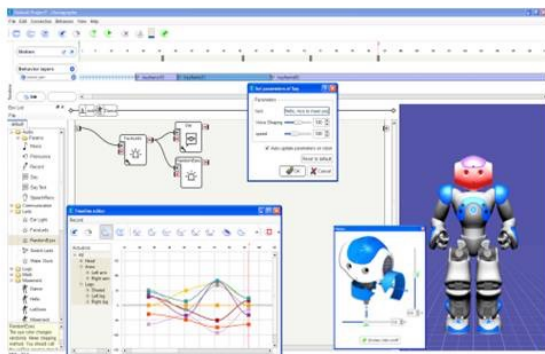


Hello everyone

RUNS ON THE COMPUTER

- » Sends orders (move, talk, ...)
- » Asks for data (image, distance, ...)

Software Suite



C Choregraphe

- ✓ Graphical Development of Behaviors
- ✓ Ergonomic and user-friendly Interface

S NAOsim

- ✓ Physical Simulation Engine
- ✓ Behaviors Simulation and validation

M Monitor

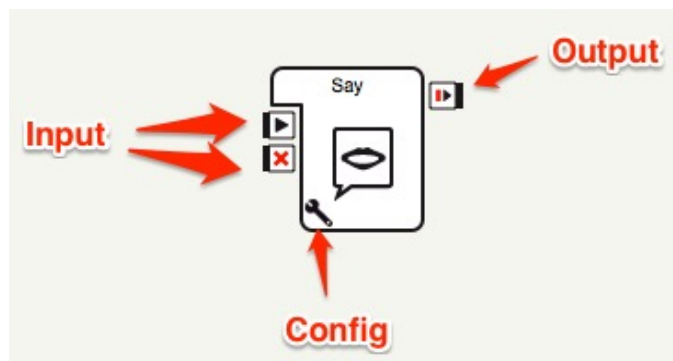
- ✓ Ergonomic Interface to monitor actuators and sensors data

SDK SDK

- ✓ Compilation and debugging tools
- ✓ MatLab, Java, Python, C++, .NET, MS Robotics Studio

Choreographe

- E' un'interfaccia grafica per la programmazione di NAO (disponibile per Windows, Mac e Linux)
(scaricabile creando un account gratuito su [Aldebaran community](#))
- E' composto da box che contengono codice per azioni specifiche



- Codice Python nei box
- Un box può essere composto da più box
- Un box può avere più inputs/outputs

Python Box

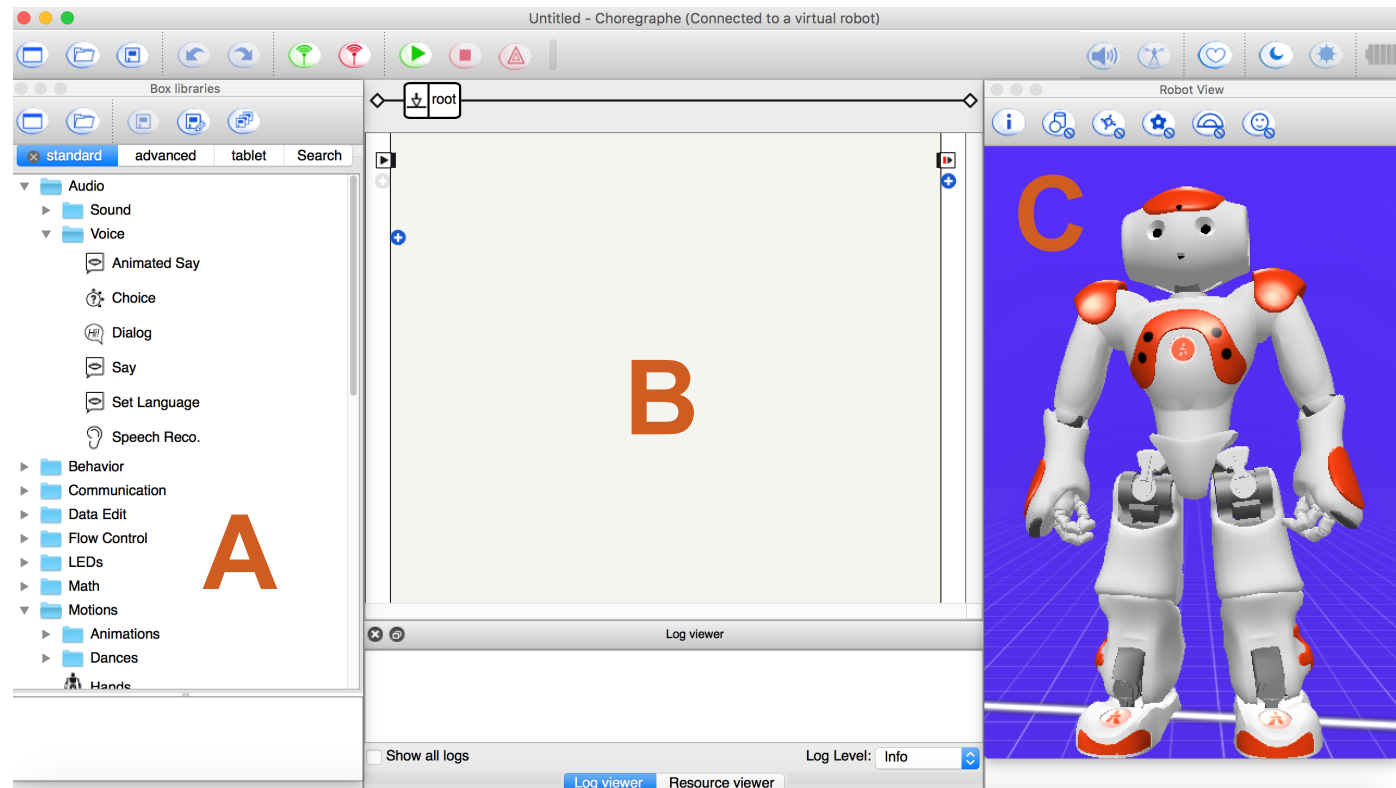
The screenshot displays the Python Box software interface. On the left, there is a 'Box libraries' panel with categories like Audio, Behavior, Communication, etc. Below it is the 'Project content' panel showing a project named 'AllenmentoLucy' with sub-projects like 'behavior_1' and 'ExampleDialog'. The main area is a 'Script editor' window titled 'Move To' containing the following Python code:

```
2 class MyClass(GeneratedClass):
3     def __init__(self):
4         GeneratedClass.__init__(self, False)
5         self.motion = ALProxy("ALMotion")
6         self.positionErrorThresholdPos = 0.01
7         self.positionErrorThresholdAng = 0.03
8
9     def onLoad(self):
10        pass
11
12    def onUnload(self):
13        self.motion.moveToward(0.0, 0.0, 0.0)
14
15    def onInput_onStart(self):
16        import almath
17        # The command position estimation will be set to the sensor position
18        # when the robot starts moving, so we use sensors first and commands later.
19        initPosition = almath.Pose2D(self.motion.getRobotPosition(True))
20        targetDistance = almath.Pose2D(self.getParameter("Distance X (m)",
21            self.getParameter("Distance Y (m)",
22            self.getParameter("Theta (deg)") * almath.PI / 180)
23        expectedEndPosition = initPosition * targetDistance
24        enableArms = self.getParameter("Arms movement enabled")
25        self.motion.setMoveArmsEnabled(enableArms, enableArms)
26        self.motion.moveTo(self.getParameter("Distance X (m)",
27            self.getParameter("Distance Y (m)",
28            self.getParameter("Theta (deg)") * almath.PI / 180)
29
30        # The move is finished so output
31        realEndPosition = almath.Pose2D(self.motion.getRobotPosition(False))
32        positionError = realEndPosition.diff(expectedEndPosition)
33        positionError.theta = almath.modulo2PI(positionError.theta)
34        if (abs(positionError.x) < self.positionErrorThresholdPos
35            and abs(positionError.y) < self.positionErrorThresholdPos
36            and abs(positionError.theta) < self.positionErrorThresholdAng):
37            self.onArrivedAtDestination()
38        else:
39            self.onStoppedBeforeArriving(positionError.toVector())
```

A purple arrow points to the 'Move To' button in the script editor's title bar. On the right side of the interface, there is a 'Pose library' panel and a 'Robot View' window showing a 3D model of a robot.

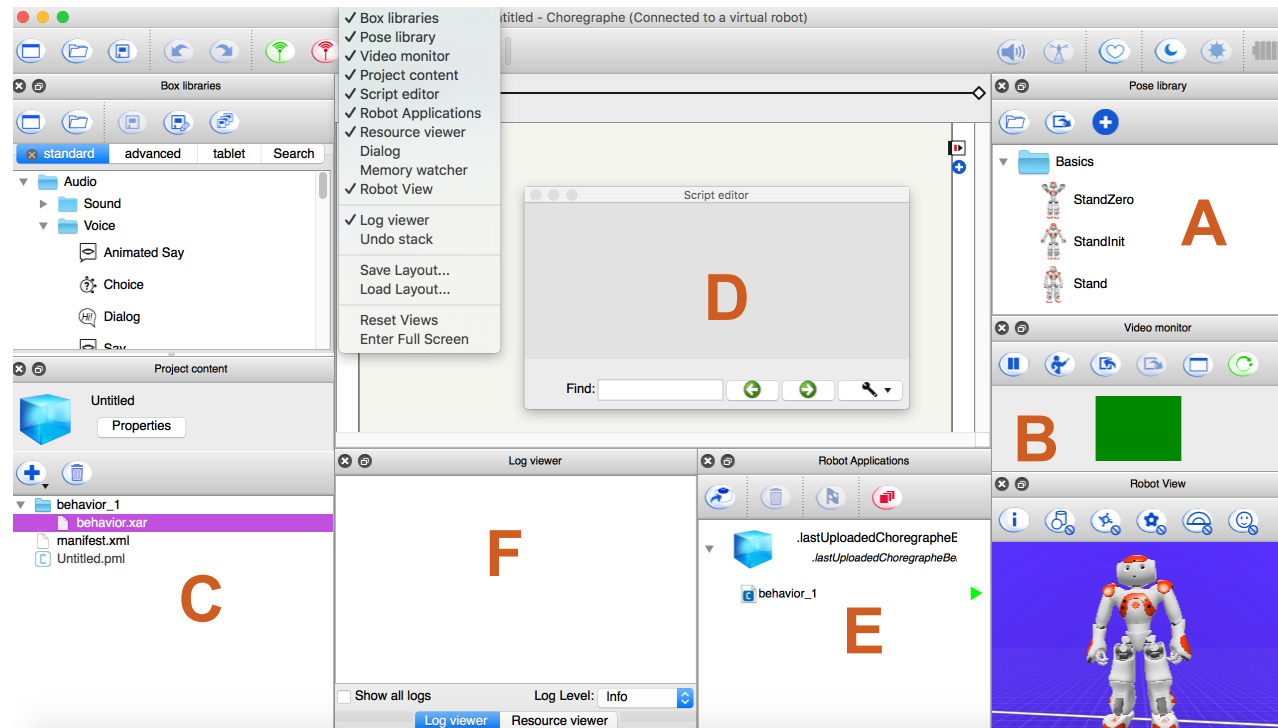
Choreographe - Panelli

- A** Box libraries panel
- B** Flow diagram panel
- C** 3D Robot View



Choregraphe - Panelli

- A** Pose library panel
- B** Video monitor panel
- C** Resource viewer panel
- D** Script editor panel
- E** Robot applications
- F** Log viewer



Demo - step 1

PiccolaProvaSal

Box libr

standard advanced tablet Search

- Audio
 - Sound
 - Voice
 - Animated Say
 - Choice
 - Dialog
 - Say**
 - Set Language
 - Speech Reco.
- Behavior
- Communication
- Data Edit
- Flow Control

onStart

Add input

Add event from ALMemory

Browse robots

| Type | Name | Port | IP |
|------|--|------|--------------|
| ★ | View web page Test LEDs Refresh IP | 9559 | 10.0.252.88 |
| ★ | | | 10.0.252.222 |
| ★ | nanoMega.local. | 9559 | 10.0.252.186 |
| ★ | nanimator.local. | 9559 | 10.0.253.16 |
| ★ | atomvince2.local. | 9559 | 10.0.252.121 |
| ★ | Nyao.local. | 9559 | 10.0.252.136 |
| ★ | WalterBishop.local. | 9559 | 10.0.252.71 |

Use fixed port: 9559

Use fixed IP/hostname: 10.0.252.185

Cancel Connect to

Demo - step 2

The screenshot displays a software interface for robot programming, likely ROS-based. The interface is divided into several panels:

- Box libraries:** A panel on the left showing a list of actions. The 'Sit Down' action is highlighted in purple. Below the list, a description reads: "Sit Down the robot tries to sit down from any position for a number of tries. Note: The number of tries can be set in parameters."
- Workspace:** A central area where a 'Sit Down' block is being added to a 'root' container. A large purple arrow points from the 'Sit Down' block in the 'Box libraries' panel to the block in the workspace.
- Pose library:** A panel on the right showing a list of poses under the 'Basics' folder: 'StandZero', 'StandInit', and 'Stand'.
- Robot View:** A window at the bottom right showing a 3D model of a robot (ARMAR-III) in a blue environment.
- Log viewer:** A window at the bottom center showing log output.

Demo - step 3

The screenshot shows a software interface for programming robot behaviors. The main workspace displays a sequence of two behavior blocks: 'Sit Down' followed by 'Stand Up', connected by a line. A large purple arrow points to the 'Stand Up' block. The 'Box libraries' panel on the left lists various behaviors, with 'Stand Up' selected. The 'Pose library' panel on the right shows a folder named 'Basics' containing three poses: 'StandZero', 'StandInit', and 'Stand'. The 'Robot View' window at the bottom right shows a 3D model of a robot. The 'Log viewer' at the bottom is currently empty.

Demo - step 4

The screenshot displays a software interface for robot animation, likely ROS Behavior Tree (BT) editor. The interface is divided into several panels:

- Box libraries:** A sidebar on the left containing various animation categories such as Communication, Data Edit, Flow Control, LEDs, Math, and Motions. Under Motions, there is an Animations folder containing 'Hello' and 'Wipe Forehead'.
- Behavior Tree:** The central workspace shows a sequence of three animation nodes: 'Sit Down', 'Stand Up', and 'Hello'. Each node contains a small robot icon in a specific pose. A large purple arrow points to the 'Stand Up' node.
- Pose library:** A sidebar on the right showing a 'Basics' folder with three poses: 'StandZero', 'StandInit', and 'Stand'.
- Robot View:** A window at the bottom right showing a 3D rendering of a white and orange humanoid robot.
- Log viewer:** A window at the bottom center, currently empty.
- Project content:** A sidebar at the bottom left showing a project structure with files like 'behavior_1', 'behavior.xar', 'manifest.xml', and 'Untitled.pml'.

Demo - step 5

The screenshot displays a software interface for programming robot behavior. The interface is divided into several panels:

- Box libraries:** Located on the left, it contains a tree view with categories like Audio, Sound, Voice, and Say. The 'Say' box is highlighted in purple. Below the tree, there is a text input field for the 'Say' box with the instruction: "Say some text. Note that you must open the box to enter the text."
- Project content:** Below the libraries, it shows a project named "Untitled" with a "behavior_1" folder containing "behavior.xar", "manifest.xml", and "Untitled.pml".
- Central workspace:** A large area showing a flowchart of behavior boxes. The flow starts at a "root" box, goes to "Sit Down", then "Stand Up", then "Hello", and finally "Say". A large purple arrow points to the "Say" box.
- Pose library:** On the right, it shows a list of poses under the "Basics" folder: "StandZero", "StandInit", and "Stand".
- Robot View:** A window at the bottom right showing a 3D model of a white and red humanoid robot.
- Log viewer:** A window at the bottom center showing log output, with a "Log Level" dropdown set to "Info".

Demo - step 6

The screenshot displays a software interface for programming a robot. The main workspace shows a 'Say' block with a 'Localized Text' sub-block containing the text 'Ciao a tutti' and a 'Say Text' block. The 'Box libraries' panel on the left lists various blocks, with 'Say' highlighted. The 'Pose library' panel on the right shows a folder 'Basics' with sub-items 'StandZero', 'StandInit', and 'Stand'. The 'Robot View' panel at the bottom right shows a 3D model of a white and orange robot. The 'Log viewer' panel at the bottom center is empty, and the 'Log Level' is set to 'Info'.

Box libraries

- Choice
- Dialog
- Say**
- Set Language
- Speech Reco.
- Behavior
- Communication
- Data Edit
- Flow Control

Localized Text

Italian

Ciao a tutti

Say Text

Pose library

- Basics
 - StandZero
 - StandInit
 - Stand

Robot View

Robot Applications Robot View

Log viewer

Show all logs Log Level: Info

Demo - step 7

The screenshot displays a software interface for robot programming, likely ROS Behavior Tree (BT) editor. The main workspace shows a behavior tree with the following structure:

- root
- Sit Down (Action)
- Stand Up (Action)
- Hello (Action)
- Say (Action)

Two purple arrows point to the play button in the top toolbar and the 'Sit Down' node in the behavior tree. The interface includes several panels:

- Box libraries:** A sidebar on the left with categories like Audio, Behavior, Communication, Data Edit, Flow Control, LEDs, Math, Motions, Sensing, System, and Templates.
- Pose library:** A sidebar on the right with a 'Basics' folder containing StandZero, StandInit, and Stand.
- Robot View:** A bottom-right window showing a 3D model of a white robot with orange accents.
- Log viewer:** A bottom-center window for monitoring logs, currently showing 'Log Level: Info'.
- Project content:** A bottom-left window showing the current project files, including 'behavior_1' and 'behavior.xar'.

Demo - step 8

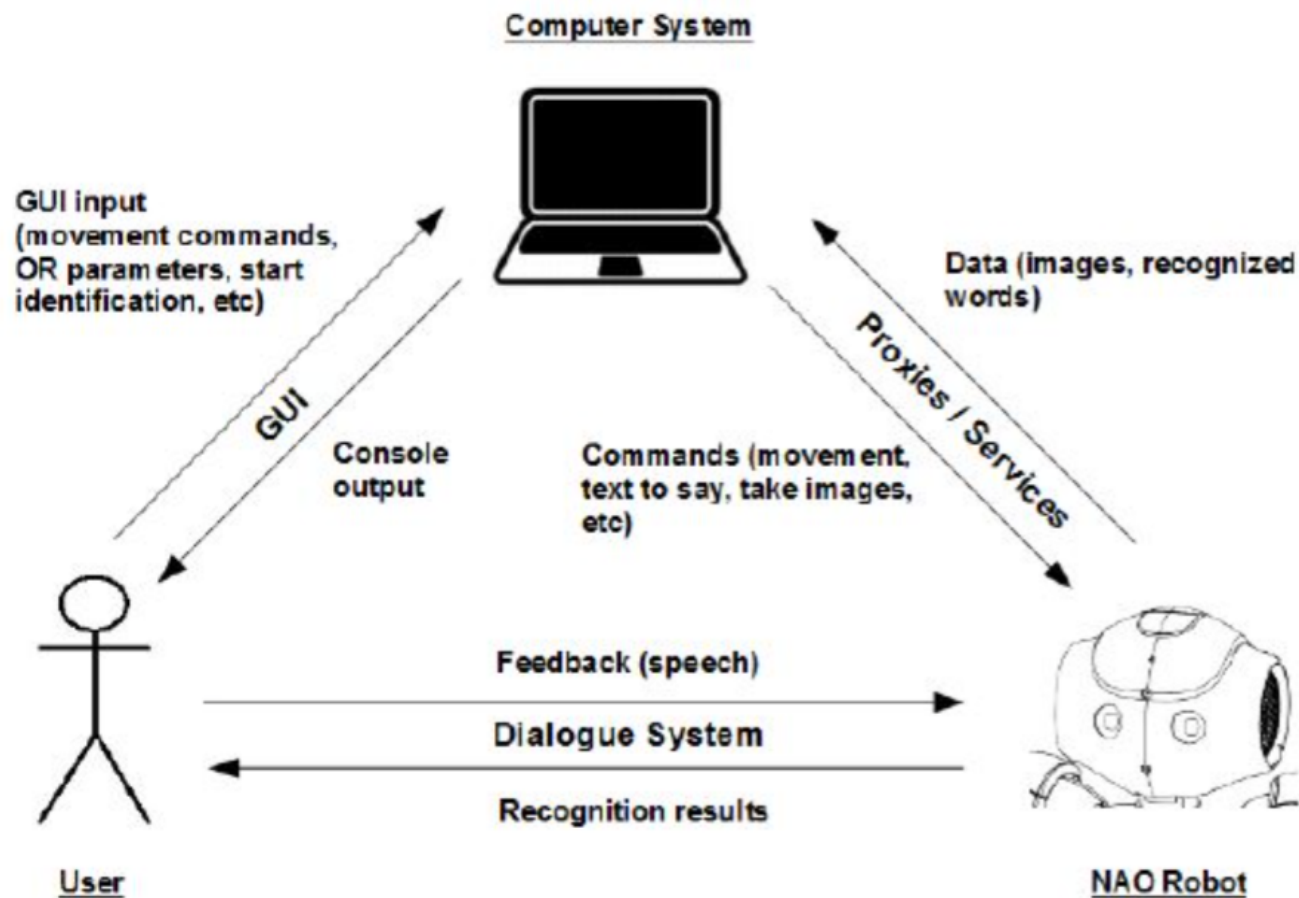
The screenshot displays a software interface for robot programming, divided into several panels:

- Box libraries:** A sidebar on the left containing various categories such as Audio, Behavior, Communication, Data Edit, Flow Control, LEDs, Math, Motions, Sensing, System, and Templates.
- Behavior Tree:** The central workspace shows a sequence of actions: "onStart" triggers "Add event from ALMemory", which leads to "Sit Down", then "Stand Up", and finally "Hello" and "Say".
- Pose library:** A panel on the right showing a "StandZero" pose under the "Basics" folder.
- Robot View:** A 3D visualization of a robot on a blue grid floor. A speech bubble above the robot says "Ciao a tutti!". A pink arrow points to the robot.
- Log viewer:** A panel at the bottom showing "Log Level: Info" and "Show all logs" options.

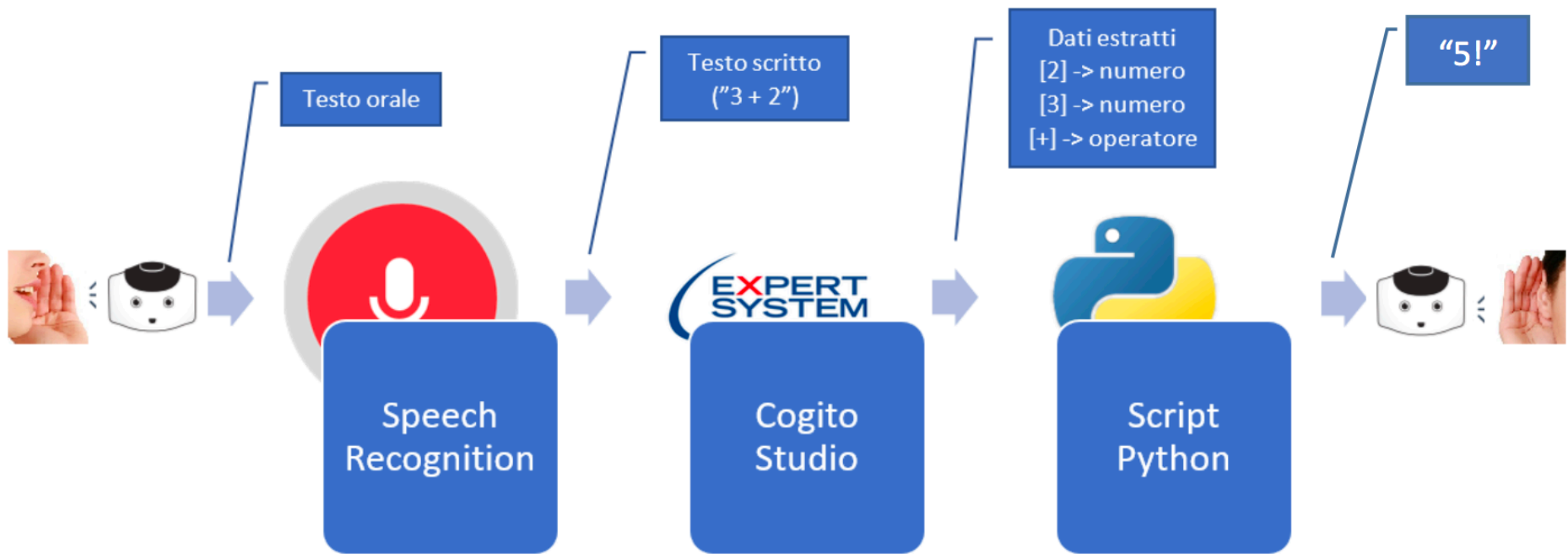
Progetti e tesi svolti

1. NAO gioca a calcio (**RoboCup**)
2. **Tesi e progetti**
 - ✓ NAO gioca a “Indovina chi”
 - ✓ Planning per azioni NAO e apprendimento di nuovi movimenti in Timeline
 - ✓ NAO si muove in una stanza
 - ✓ NAO matematico
3. **Tesi magistrali**
 - ✓ Reti neurali per
 - ▶ Riconoscimento facciale
 - ▶ OCR (Riconoscimento di caratteri)
 - ✓ Imitazione di movimenti umani tramite uso di Kinect

Riconoscimento facciale



NLP e matematica



Lucy legge con OCR

4) the robot pronounce the text according to the language installed



1a) getImageRemote command is sent to the robot



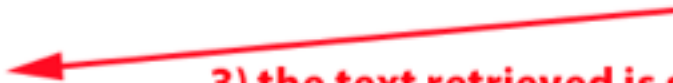
1b) the image is retrieved from the robot

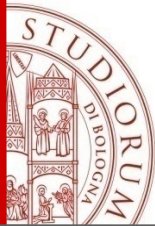


2) the image is processed by our OCR software



3) the text retrieved is sent to the robot





Proposte di progetti e tesi

1. NAO e linguaggio naturale

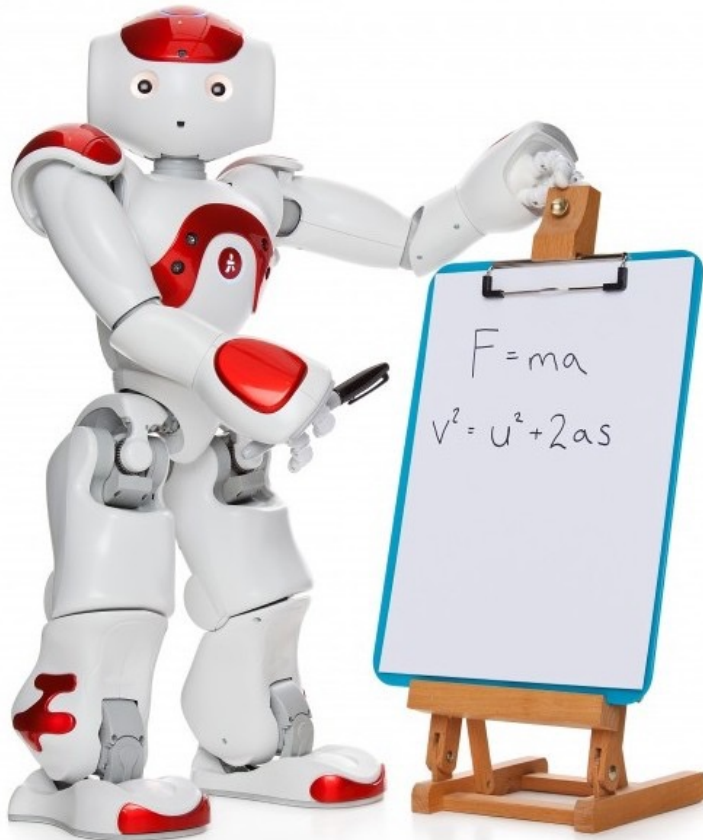
- ✓ Sviluppo di bot
- ✓ Sviluppo di giochi interattivi
- ✓ Sviluppo di abilità per quiz e giochi matematici

2. NAO e planning real time

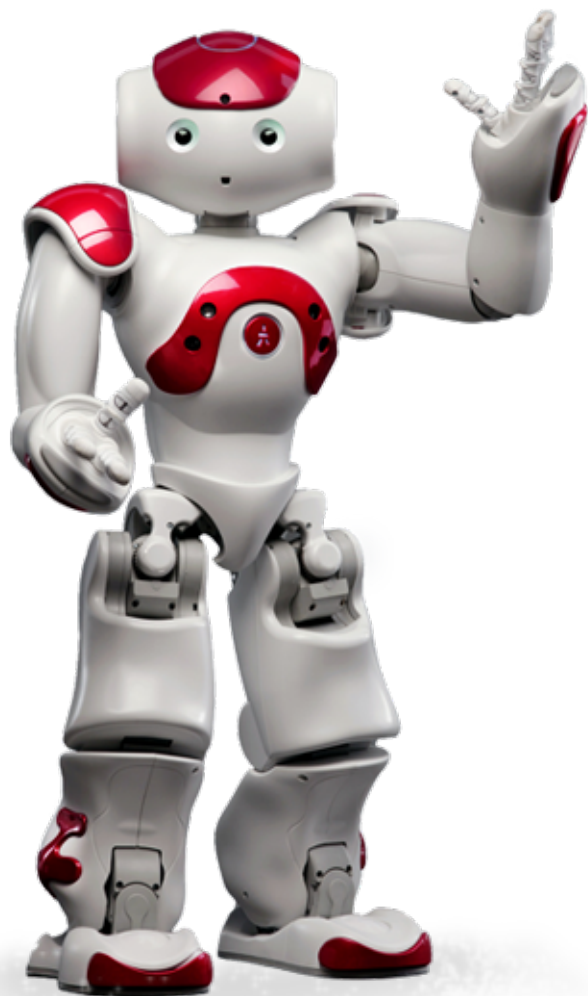
- ✓ Esecuzione di azioni (comandi) real-time in ambiente chiuso
- ✓ Movimento all'interno di un percorso

3. Sviluppo e ampliamento di progetti precedenti

Presentiamo il nostro NAO: Lucy



Lasciamo che Lucy
si presenti!



Grazie per l'attenzione!