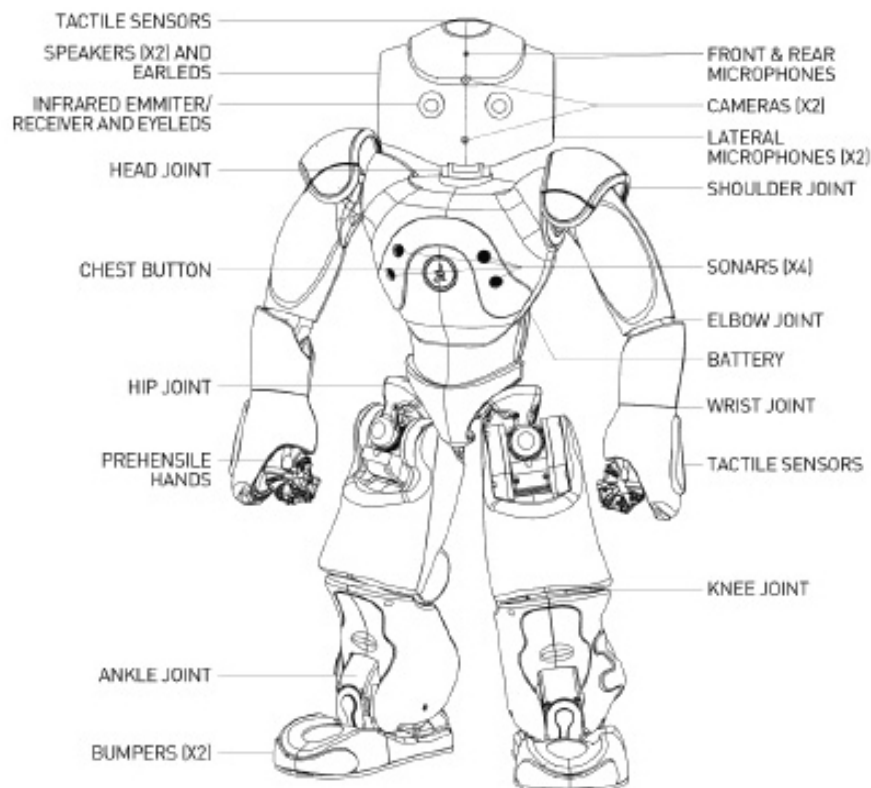


# NAO

## *Programming a humanoid robot*



# Nao at a glance



# What can NAO do?

## MOVE

- 25 degrees of freedom
- Motors controlled by software
- Complex movement capabilities

## SENSE

- 2 HD camera
- 4 microphones
- 2 bumpers
- 2 sonars

## INTERACT

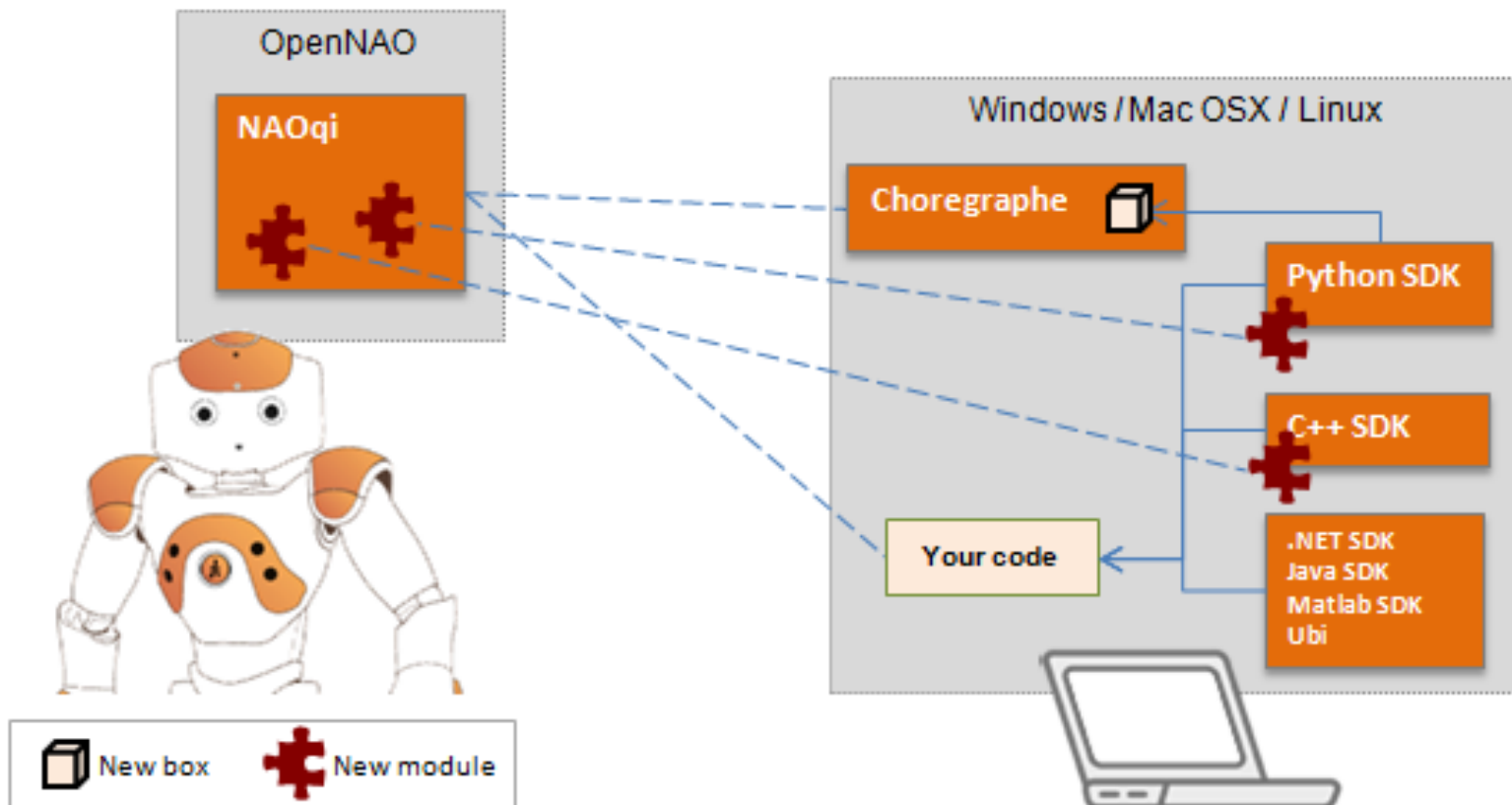
- 2 speakers
- multiple LEDs
- tactile sensors
- prensile hands
- infrared sensors
- WiFi connection

## THINK

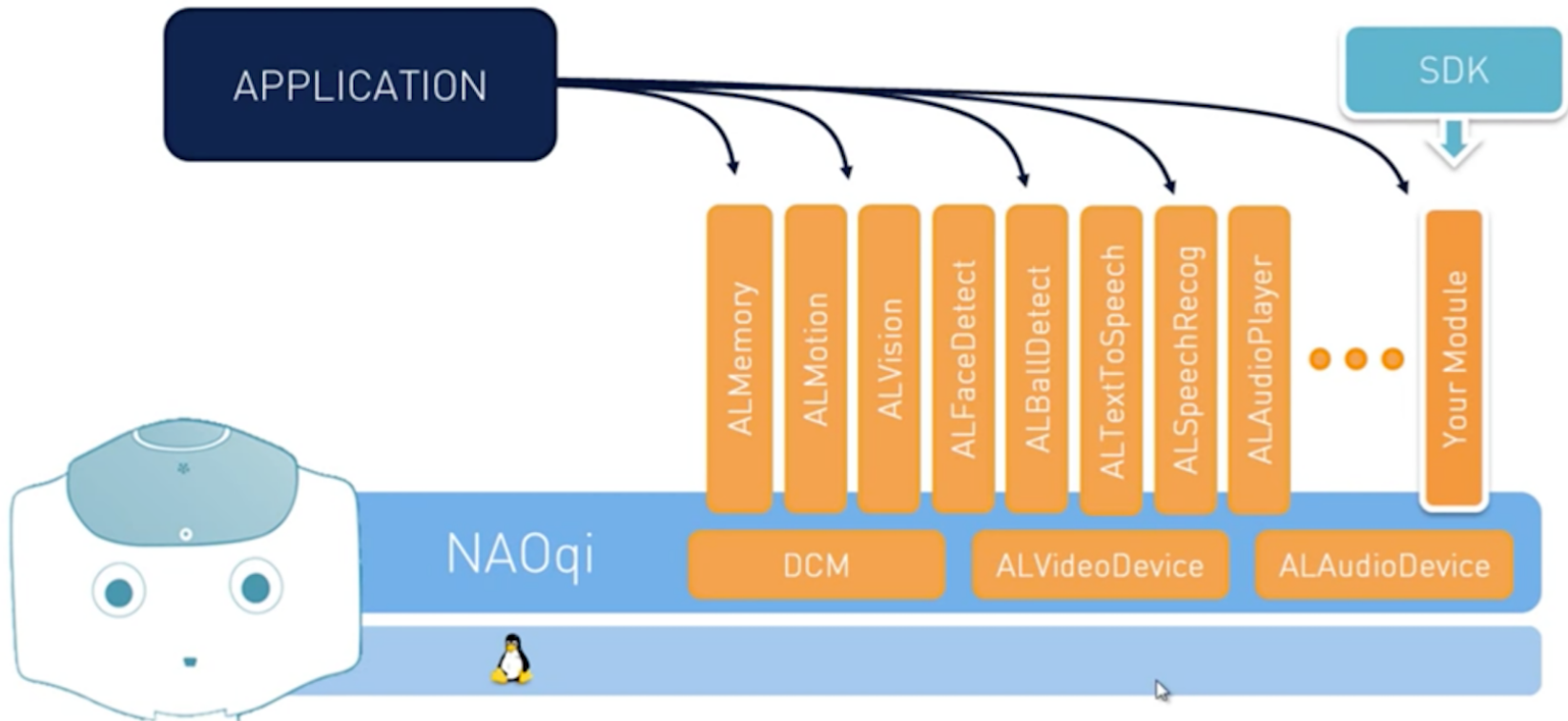
- Intel Atom 1,6 GHz CPU
- 1 Gb RAM
- 8 Gb Flash Memory
- Software suite

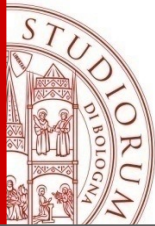


# What and where?



# Why programming in Python





# Remote control

## STANDARD APPROACH:

1. Import ALProxy

```
from naoqi import ALProxy
```

2. Create an ALProxy to the module you want to use

```
tts = ALProxy( "ALTextToSpeech" , "ip_address" , 9559 )
```

3. Call a method

```
tts.say( "Hello everyone! I am happy to work with you!" )
```

APPLICATION



# Remote control



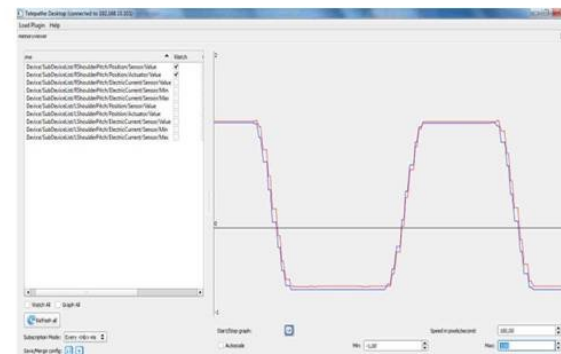
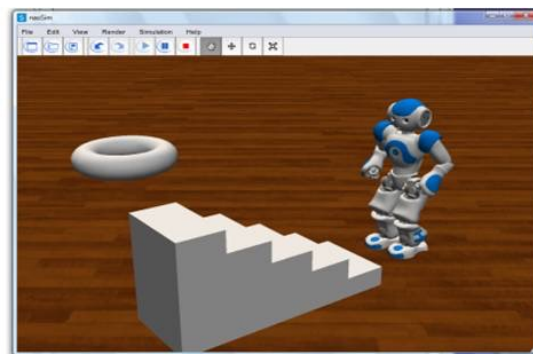
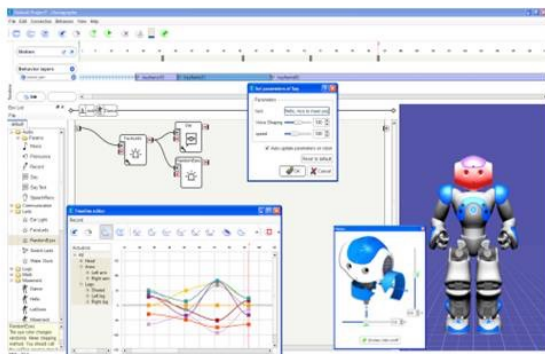
Say("Hello everyone! ...")



Hello everyone

## RUNS ON THE COMPUTER

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



## **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

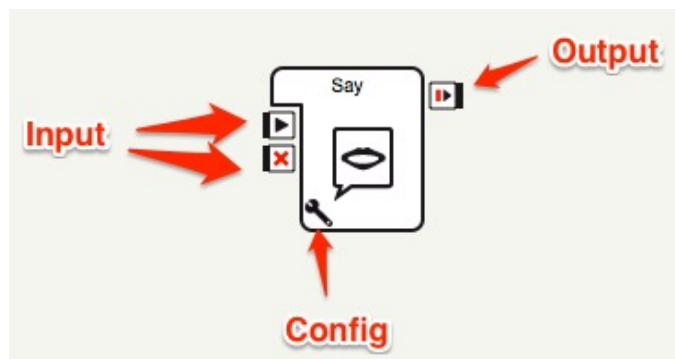
## **SDK** SDK

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



# Choreographe

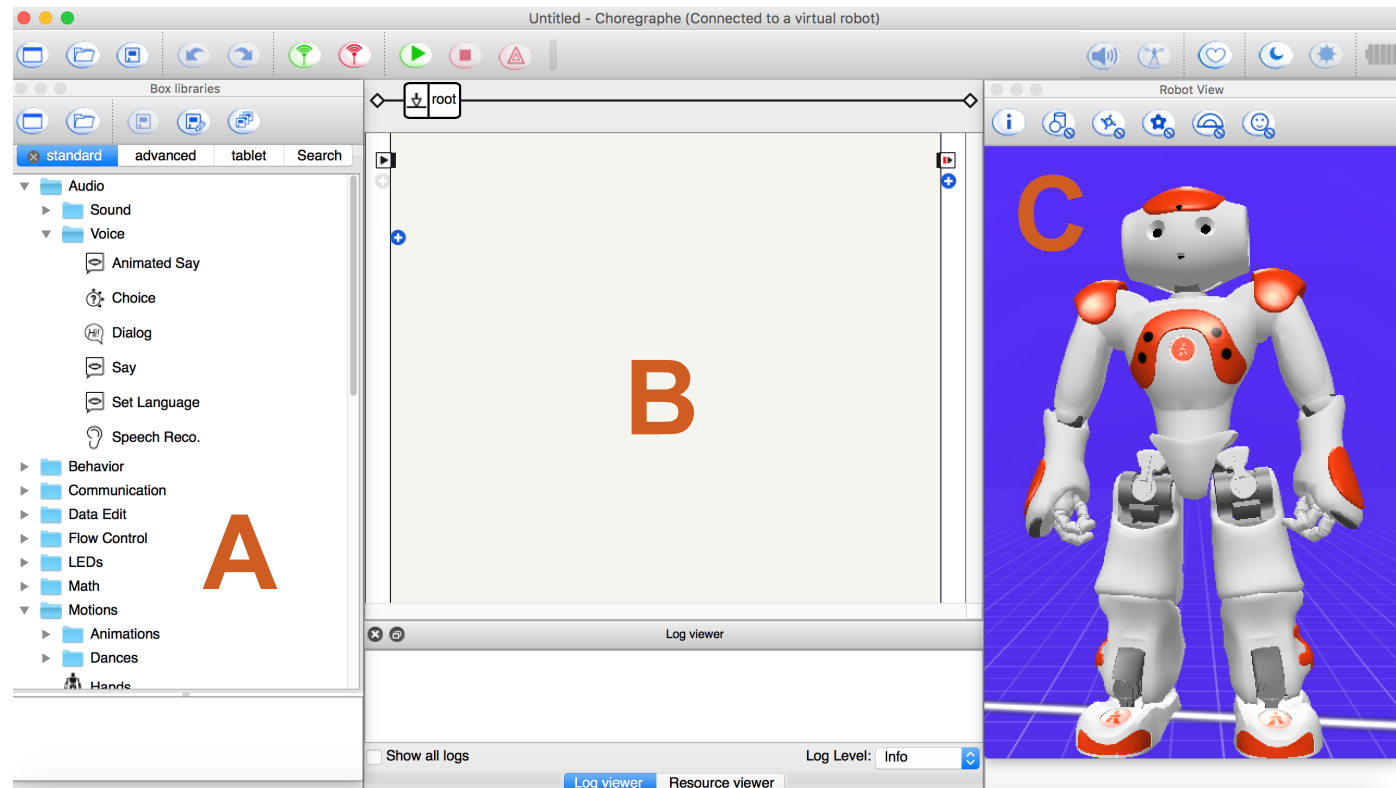
- It is a graphical interface to program NAO (for Windows, Mac e Linux)  
(downloadable after creating an account on [Aldebaran community](#))
- It is composed by boxes containing some code for specific actions

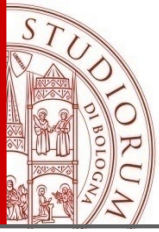


- Python code into boxes
- a box can be made of other boxes
- a box can have different inputs/outputs

# Choreographe - Panels

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





# Sample Demo

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

- Box libraries:** A sidebar on the left containing a tree view of categories such as Audio, Behavior, Communication, Data Edit, Flow Control, LEDs, Math, Motions, Sensing, System, and Templates. The 'Audio' category is currently selected.
- Central workspace:** A diagram showing a sequence of behavior nodes: 'Sit Down' (with a robot icon sitting), 'Stand Up' (with a robot icon standing), 'Hello' (with a robot icon and a speech bubble), and 'Say' (with a robot icon and a speech bubble). A play button icon is visible in the top toolbar, and a purple arrow points to it. Another purple arrow points to a plus sign in the workspace.
- Pose library:** A panel on the right showing a list of poses under the 'Basics' folder: StandZero, StandInit, and Stand. Below this is a 'Video monitor' and 'Pose library' tab.
- Robot View:** A window at the bottom right showing a 3D model of a white robot with orange accents on a blue grid background. Below the robot are tabs for 'Robot Applications' and 'Robot View'.
- Log viewer:** A panel at the bottom center showing a 'Log Level' dropdown set to 'Info' and a 'Show all logs' checkbox.

# Python Boxes

The screenshot displays a software development environment with several panels:

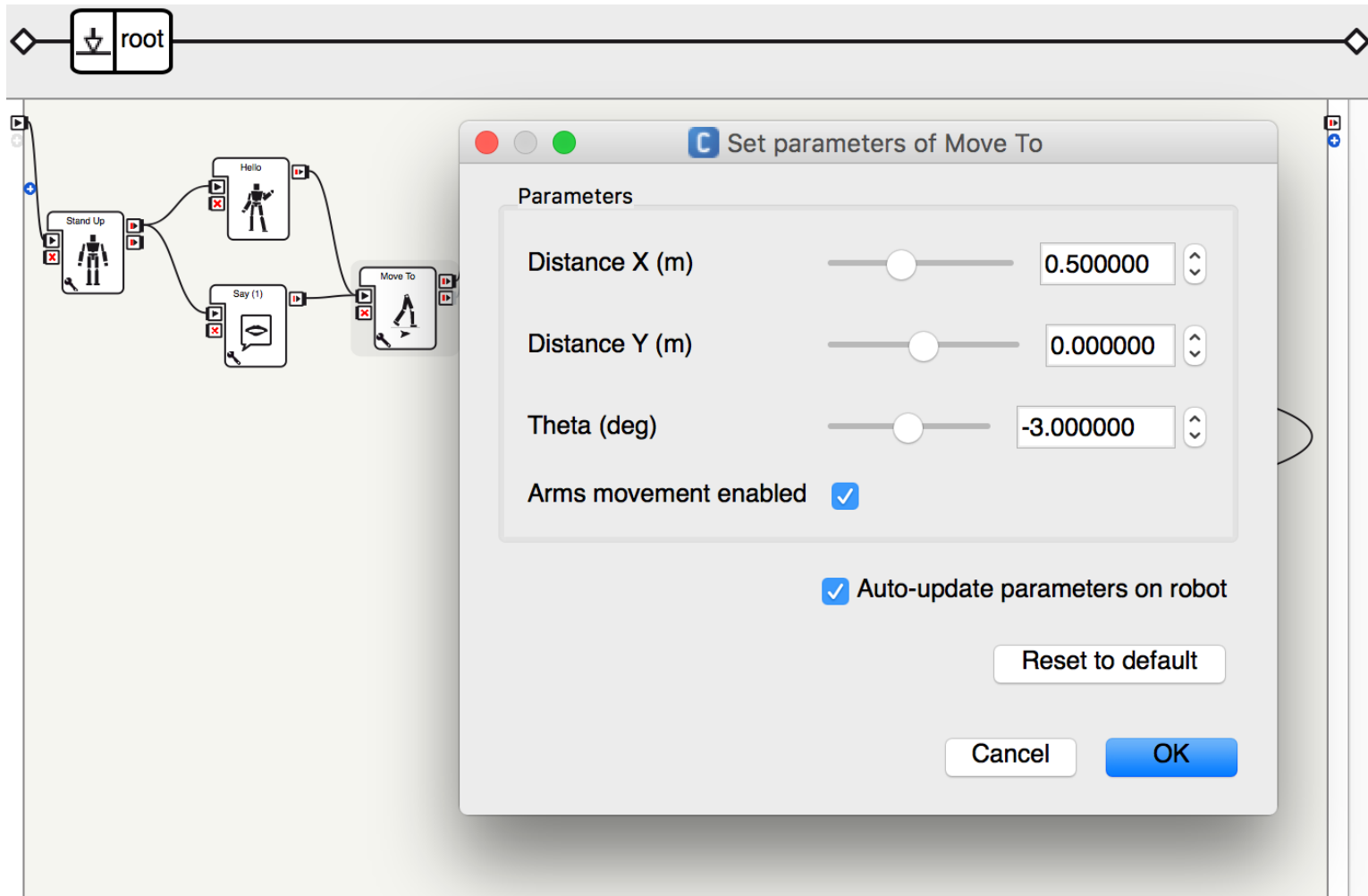
- Box libraries:** A sidebar on the left showing various categories like Audio, Behavior, Communication, etc.
- Project content:** A panel below the libraries showing a project named 'AllenmentoLucy' with sub-projects like 'behavior\_1' and 'ExampleDialog'.
- Script editor:** The central window showing a Python class named `MyClass` that inherits from `GeneratedClass`. The class includes methods for initialization, loading, unloading, and handling input to move a robot to a specific position. A purple arrow points to the `Move To` button in the script editor's toolbar.
- Robot View:** A panel on the right showing a 3D model of a robot in a virtual environment.

```

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())

```

# Configuration Box



The image shows a configuration window titled "Set parameters of Move To" overlaid on a state machine diagram. The state machine has a root node with a dropdown arrow, leading to a "Stand Up" state. From "Stand Up", there are two transitions: one to a "Hello" state and one to a "Say (1)" state. Both "Hello" and "Say (1)" have transitions leading to a "Move To" state. The "Move To" state is currently selected, and its configuration parameters are shown in the dialog box.

**Set parameters of Move To**

Parameters

- Distance X (m): 0.500000
- Distance Y (m): 0.000000
- Theta (deg): -3.000000
- Arms movement enabled:

Auto-update parameters on robot

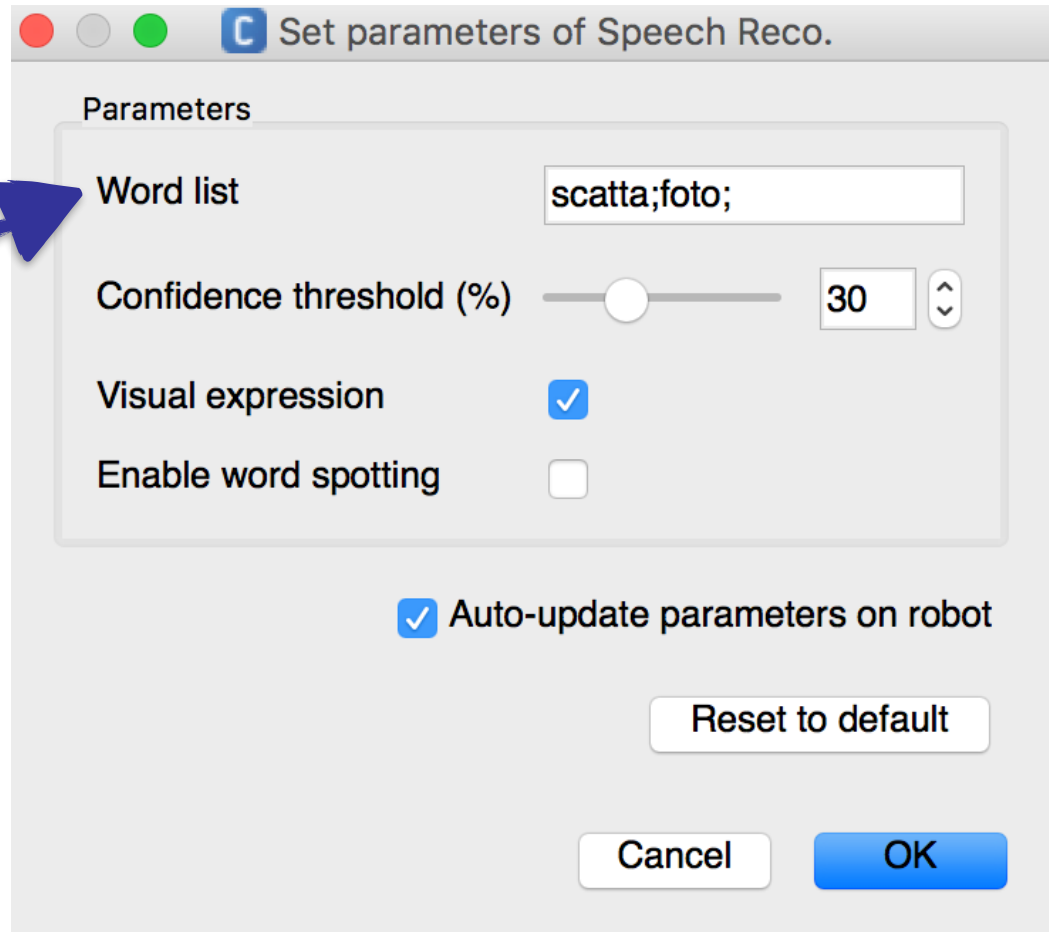
Reset to default

Cancel OK



# Speech Recognition

You can enter a list of words to be recognized by voice command to the robot



Set parameters of Speech Reco.

Parameters

Word list

Confidence threshold (%)  30

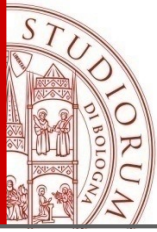
Visual expression

Enable word spotting

Auto-update parameters on robot

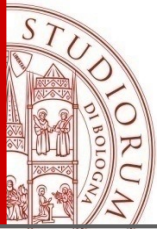
Reset to default

Cancel OK



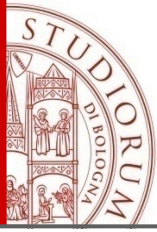
# Projects and Thesis

1. NAO plays Soccer (**RoboCup**)
2. **Thesis and projects**
  - ✓ NAO plays “Guess who”
  - ✓ Planning for NAO actions and learning new movements in Timeline
  - ✓ NAO navigates in different rooms
  - ✓ NAO mathematician
3. **Master Thesis**
  - ✓ Neural networks for
    - ▶ Face recognition
    - ▶ OCR (Optical Character Recognition)
  - ✓ User movements imitation using Kinect



# Nao plays Soccer

- RoboCup aims to create, by 2050, a **team of humanoid robots** that can take on and beat the best human players.
- When playing together, the **robots must act autonomously** and are unable to get help from their handlers.
- They also communicate via wi-fi to **co-ordinate teamwork**.
- **Various technologies** have to be developed in **AI**: the robots know who to pass to and how best to defeat an opponent.



# Lucy plays "Guess who"

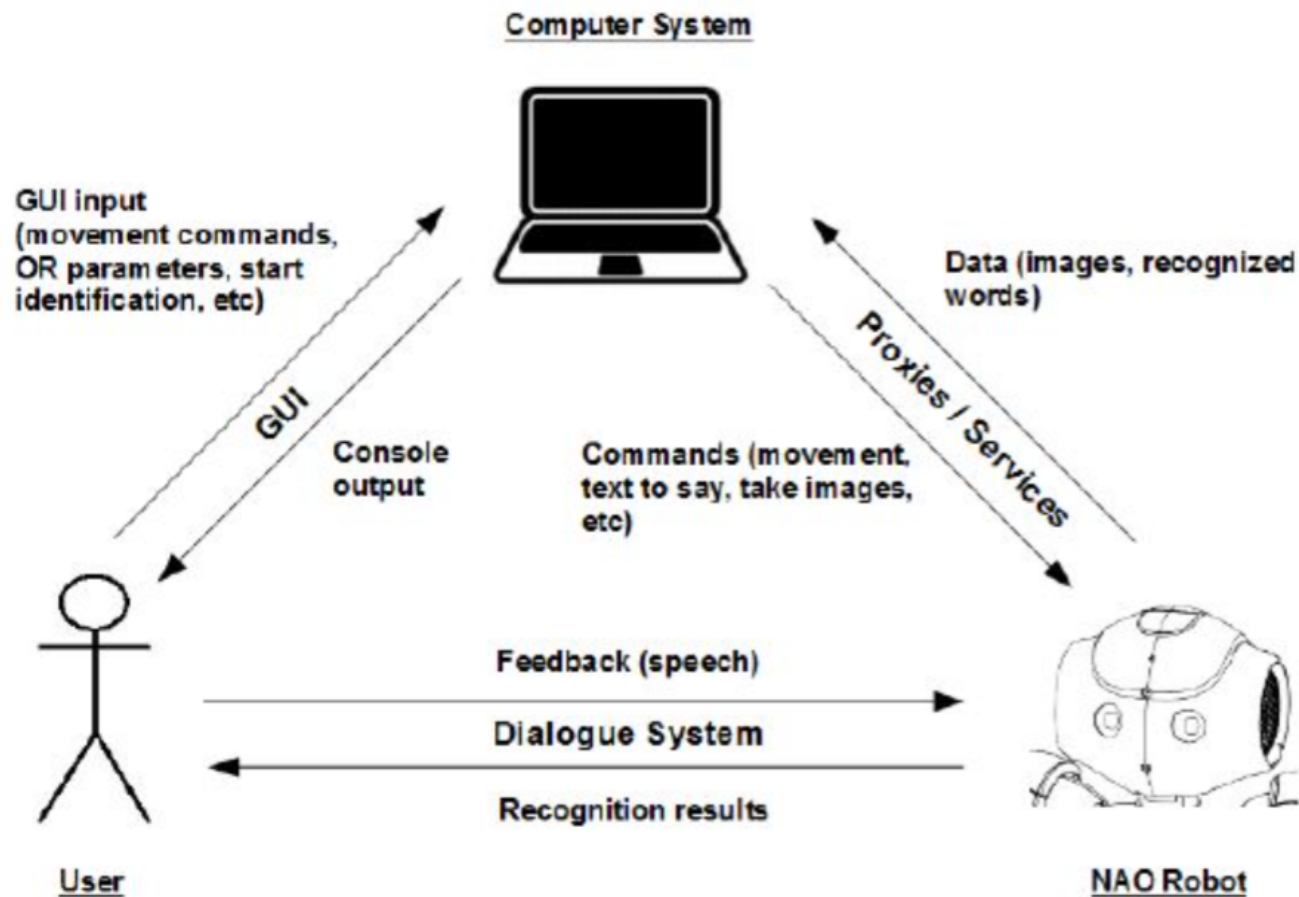
---

- Lucy is **able to play guess who with a human** using voice recognition.
- Once the sentence pronounced is translated into textual form, **Lucy is able to understand** what was communicated



## Natural Language Processing

# Face Recognition





# Lucy reads with 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**



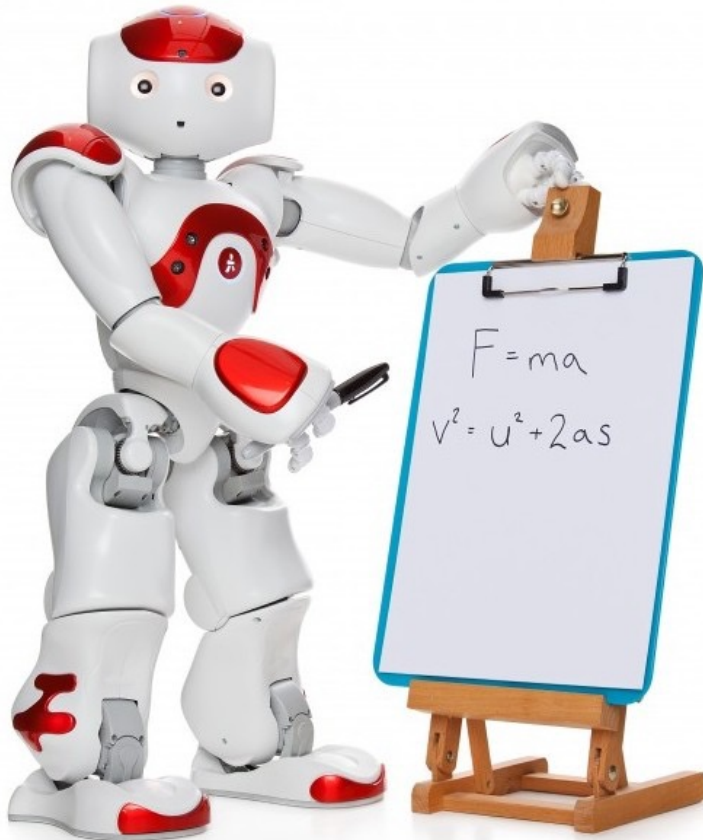
**2) the image is processed by our OCR software**



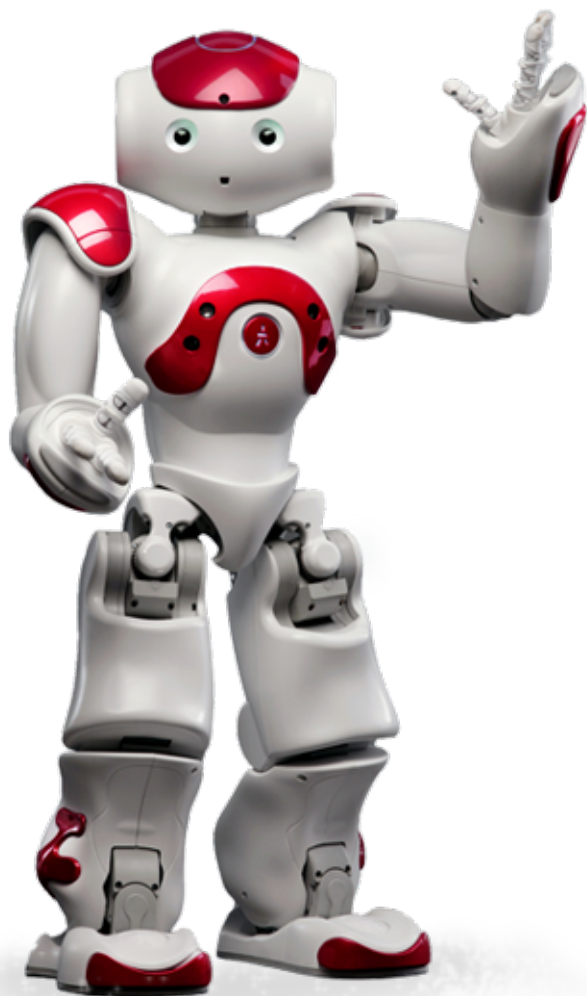
**3) the text retrieved is sent to the robot**



# Presenting our NAO: Lucy



Let Lucy  
introduce herself!



# Thank you!