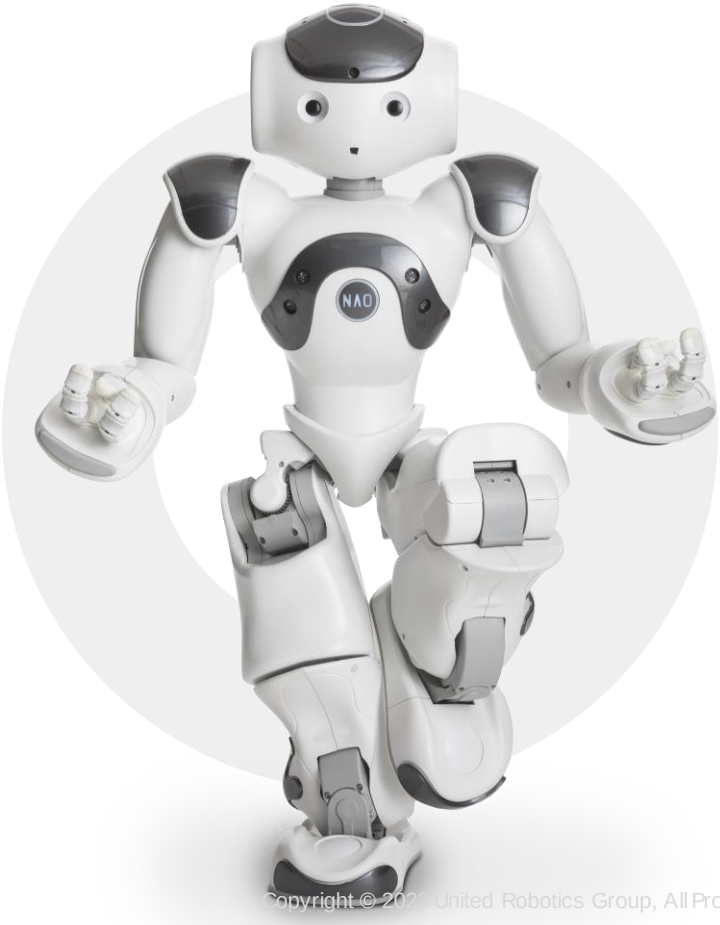
NAO6

NAO Challenge Training

Let's make NAO move

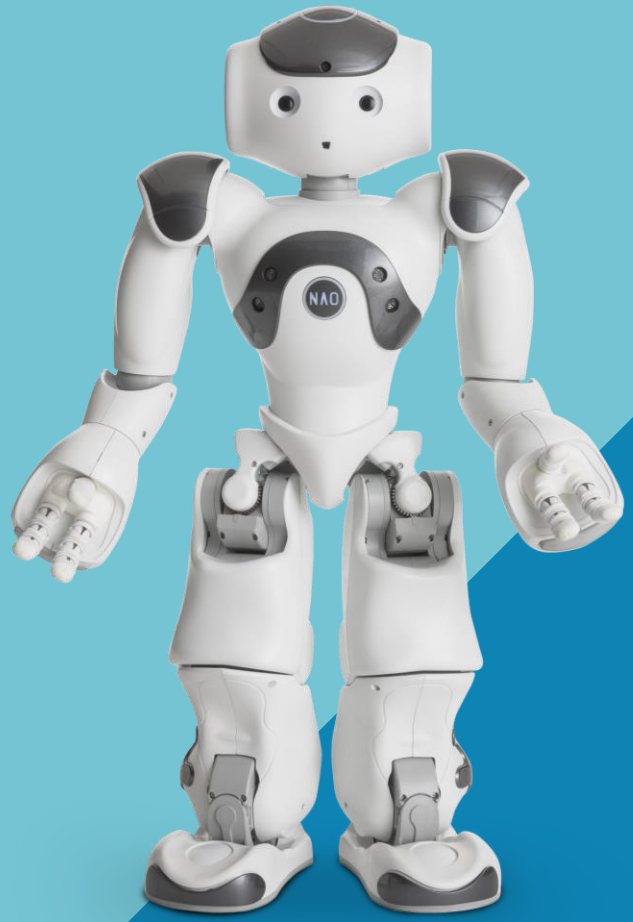
September 2019



Welcome!

- Using NAO
  - Discovering the hardware
  - Choregraphe
  - Python
- NAO Challenge

NAO Hardware



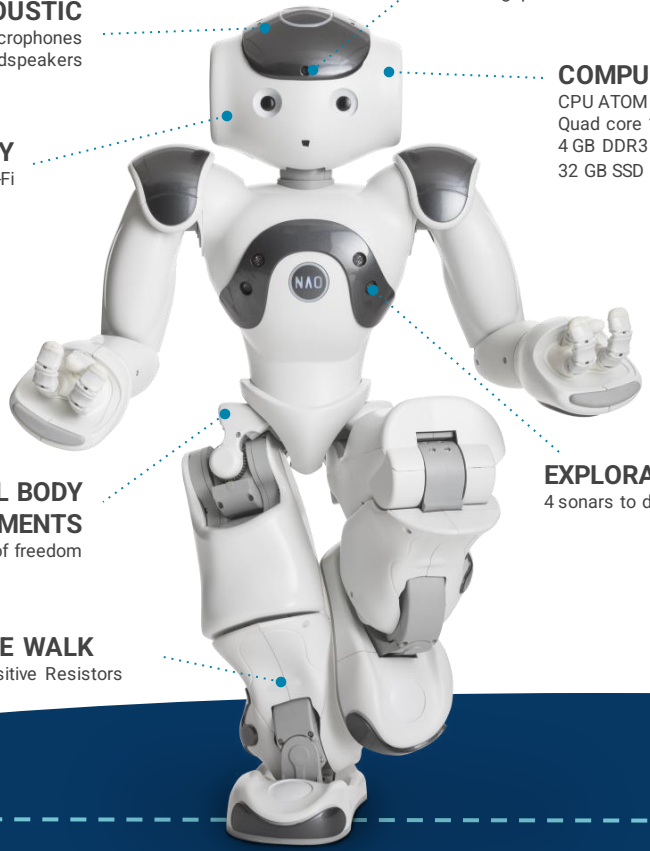
THE 6TH GENERATION OF HUMANOID ROBOT NAO

ACOUSTIC
4 omnidirectional microphones
2 loudspeakers

DETECTION
2 5-megapixels cameras

COMPUTING
CPU ATOM E3845
Quad core 1.91 GHz
4 GB DDR3 RAM
32 GB SSD

CONNECTIVITY
Bluetooth – Ethernet - Wi-Fi



58 cm

EXPLORATION
4 sonars to detect obstacles

GRACEFUL BODY MOVEMENTS
25 degrees of freedom

ADAPTIVE WALK
8x Force Sensitive Resistors

5,5 kg

NAO6

SPEECH RECOGNITION
More than 20 Languages

FALL MANAGER
Detects falls & trigger the protection

FALL RECOVERY
Able to stand up alone

NAO^6, 6 MAJOR AREAS OF HARDWARE IMPROVEMENT

1 - **POWER**
CPU – ATOM Z530 E3845
Memory: 8GB SDCard 32GB SSD

2 - **VISION**
Camera – Aptina MT114 1.3Mpix Omnivision 5640 5Mpix

3 – **AUDIO**
Cardioid mic Omnidirectional mic + Audio codec

4 – **MOTION**
Motors - PortEscape 16 GT PortEscape 16 GT MP

5 – **ROBUSTNESS**
Finger - Robustness Improvement

6 – **CONNECTIVITY**
Bluetooth
New Wi-Fi
Better Ethernet

1 - POWER
NAO^6 benefits from the progress made on Pepper which makes it more powerful.

2 - VISION
With its new cameras, NAO^6 better detects people. Dual stream of the top and bottom cameras is now supported.

3 - AUDIO
NAO^6 hears and understands a lot better making dialog a full part of the interactions.

4 - MOTION
Thanks to its new motors, NAO^6 can move longer without overheating.

5 - ROBUSTNESS
NAO^6 benefits from many robustness improvements on motors and fingers providing an extended lifetime.

6 - CONNECTIVITY
NAO^6 is more connected than ever with Bluetooth and a more efficient and faster Wi-Fi.

NAO6 CAN BE USED IN VARIOUS SECTORS

An **innovative and advanced interface of** communication, interaction and education

EDUCATION

- **Any type of school** from kindergarten to University

- *Coding & programing*
- *Special education*
- *Literature & learning*
- *Laboratory & Research*

HEALTHCARE

- Adapted solution for caregiver and patient, for any type of facility

- Hospital
- Specialized facility
- Nursing home
- Pharmacy

CUSTOMER FACING ENVIRONMENT

- Any **physical places** for innovative assistance & dynamic communication

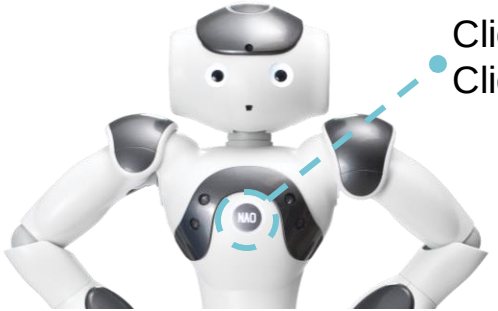
- Retail
- Travel & Hospitality
- Company & coworking space
- Bank & Assurance
- Public service

Basic orientation with NAO

[REST] <> [AWAKE] | *Changing State*

To change between Rest & Awake, double-click the chest button!

Rest

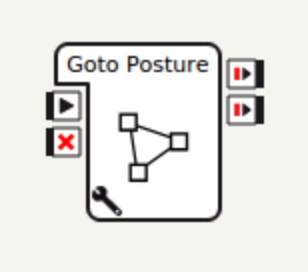


Wake up

[STAND] <> [CROUCH] / *Changing Position*

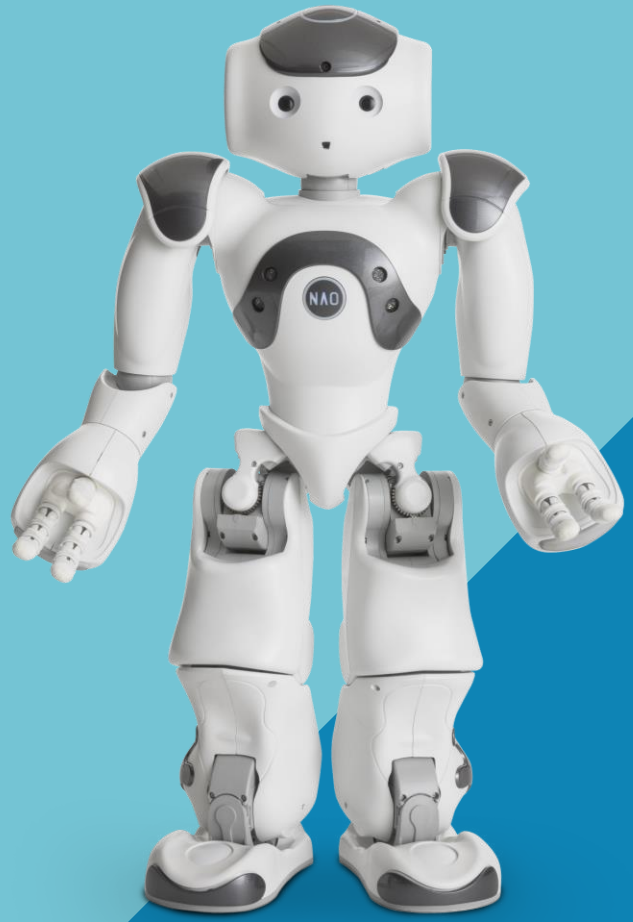
Changing state is different from changing Posture

To change between Stand & Crouch position use Choregraphe icons or Change posture box!

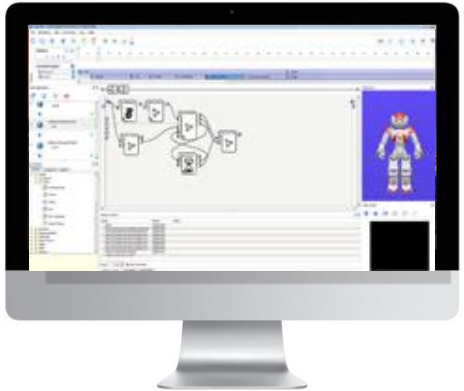


NAO Software

Topic I - Overview



Building out your Toolkit | *Software*



Choregraphe

Easy visual prototyping tool

Software Development Kit

Comprehensive API for C++ or Python

Jupyter Lite

Experimental: Python 3 without installing

The head | *Software*

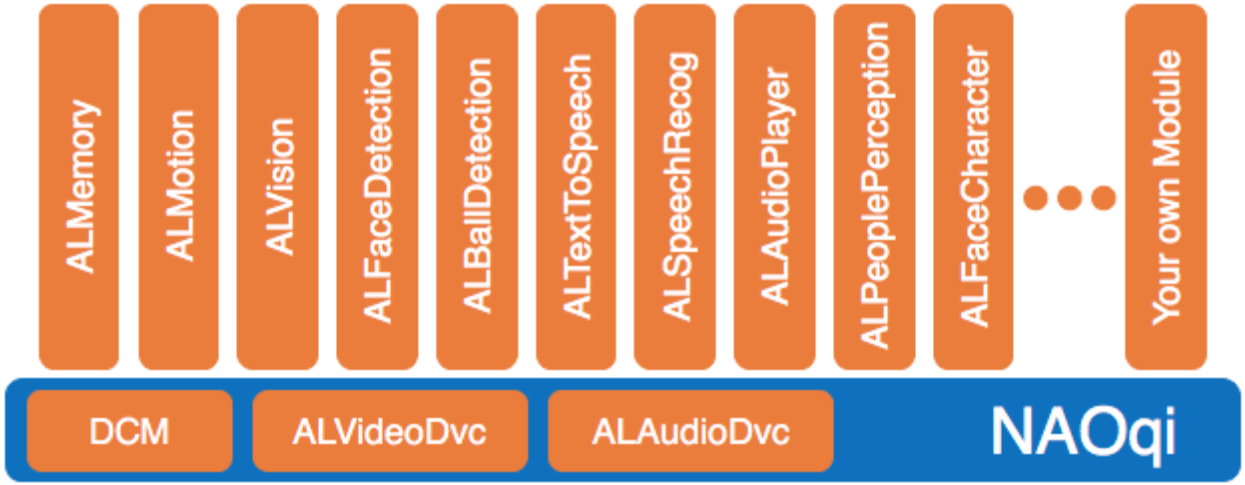
- Runs on Linux
- **Executes the program**
- Stores the data
- **Connects to the Internet**
- **Serves the app's webpages**

NAOqi + Service Architecture | *Software*

- The OS is called NAOqi
  - Softbank Robotics own robotic operating system framework
  - Running on top of the Linux operating system
  - Allows local and remote (network) access to Nao's APIs
  - Executes Choregraphe behaviours and Python code

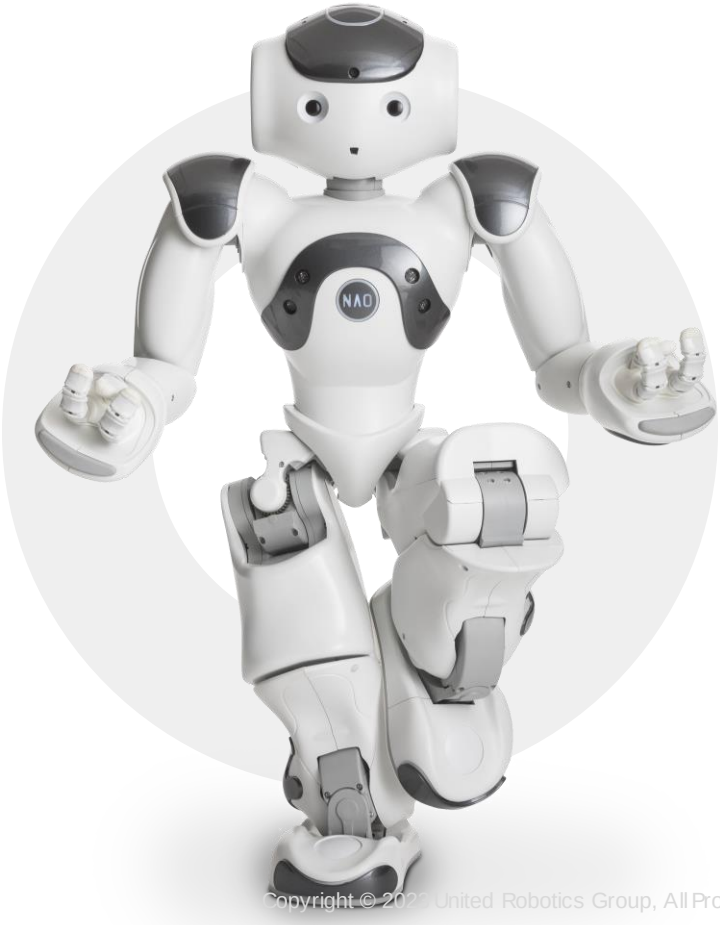
NAOqi + Service Architecture | *Software*

- NAOqi works with services



Software - Topic II

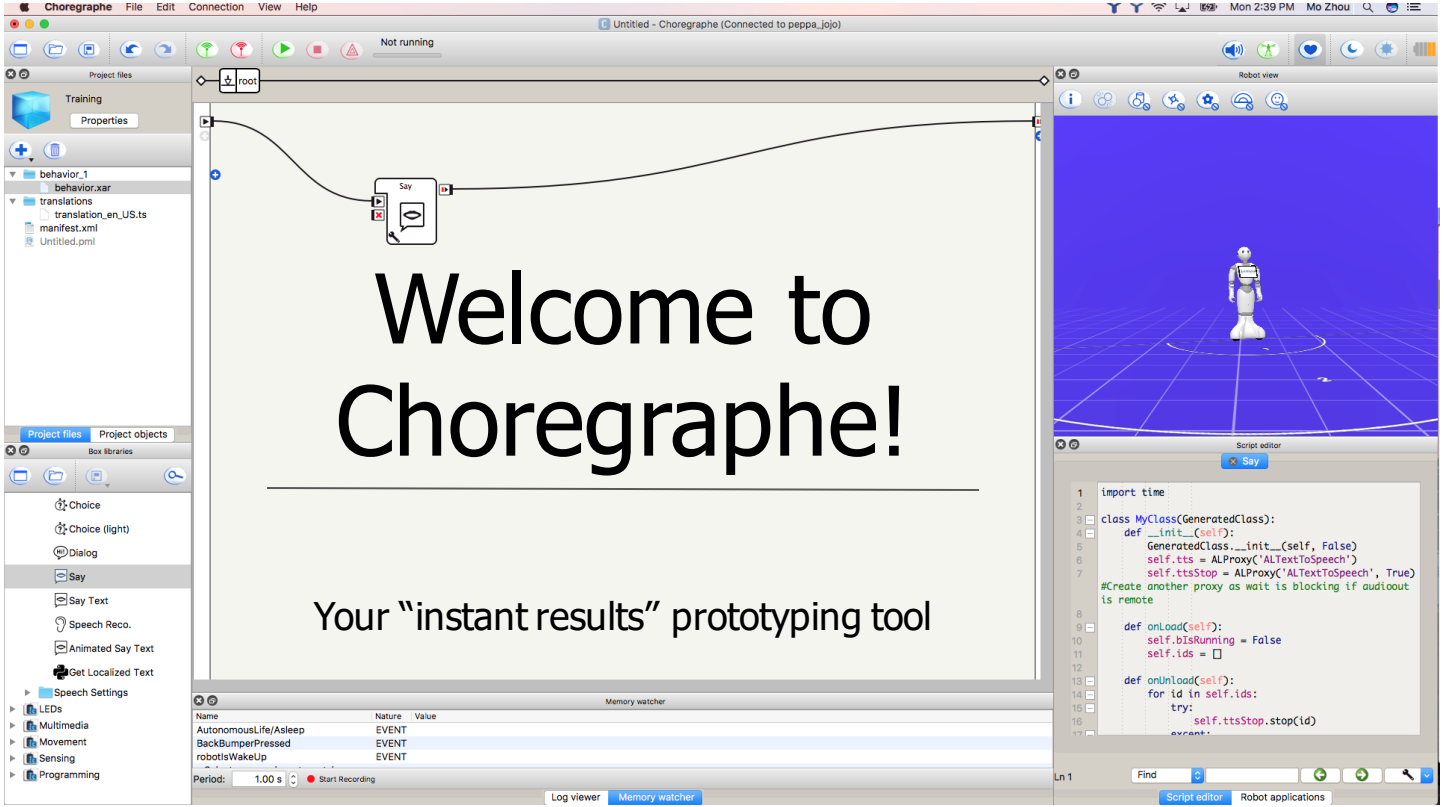
Choregraphe

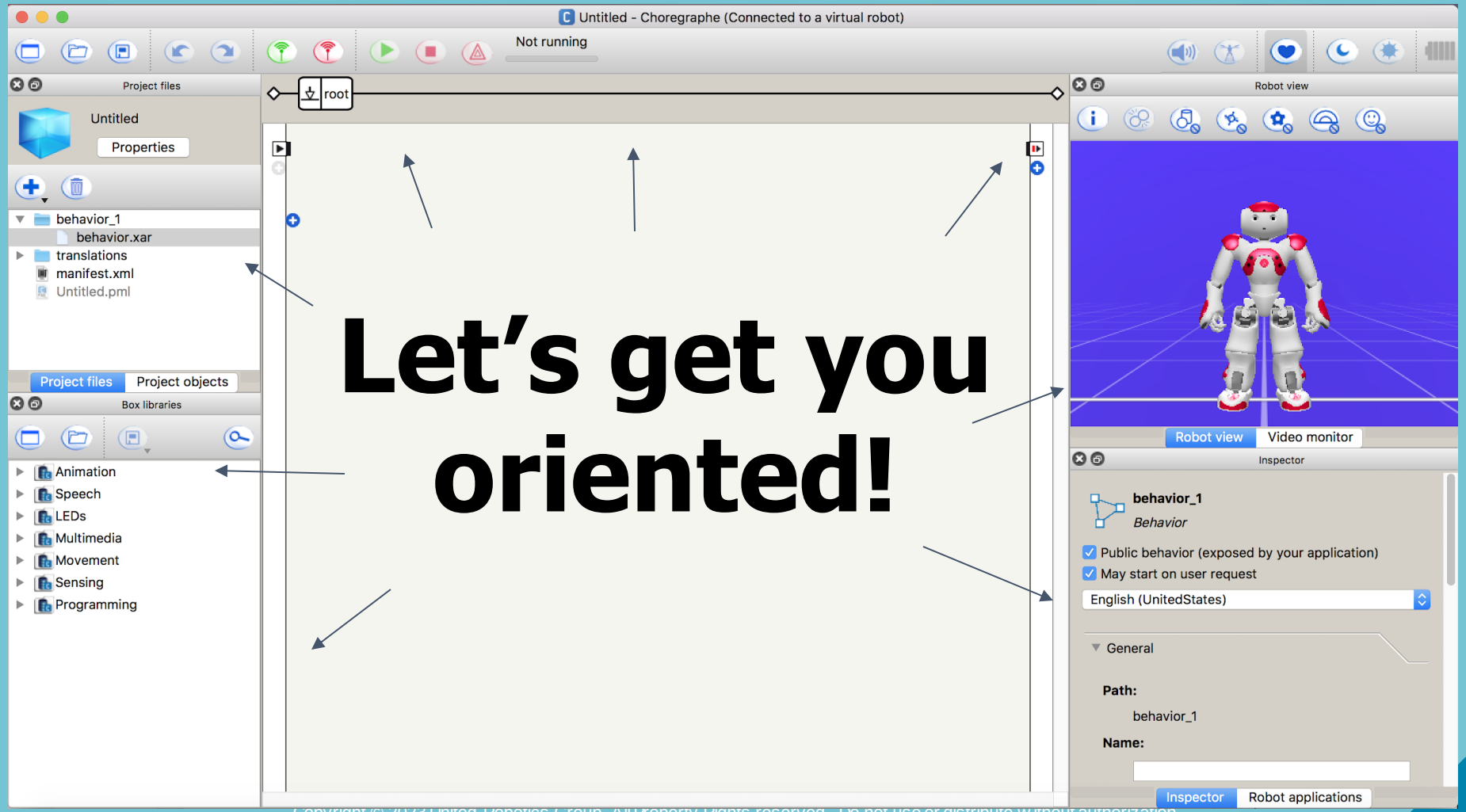


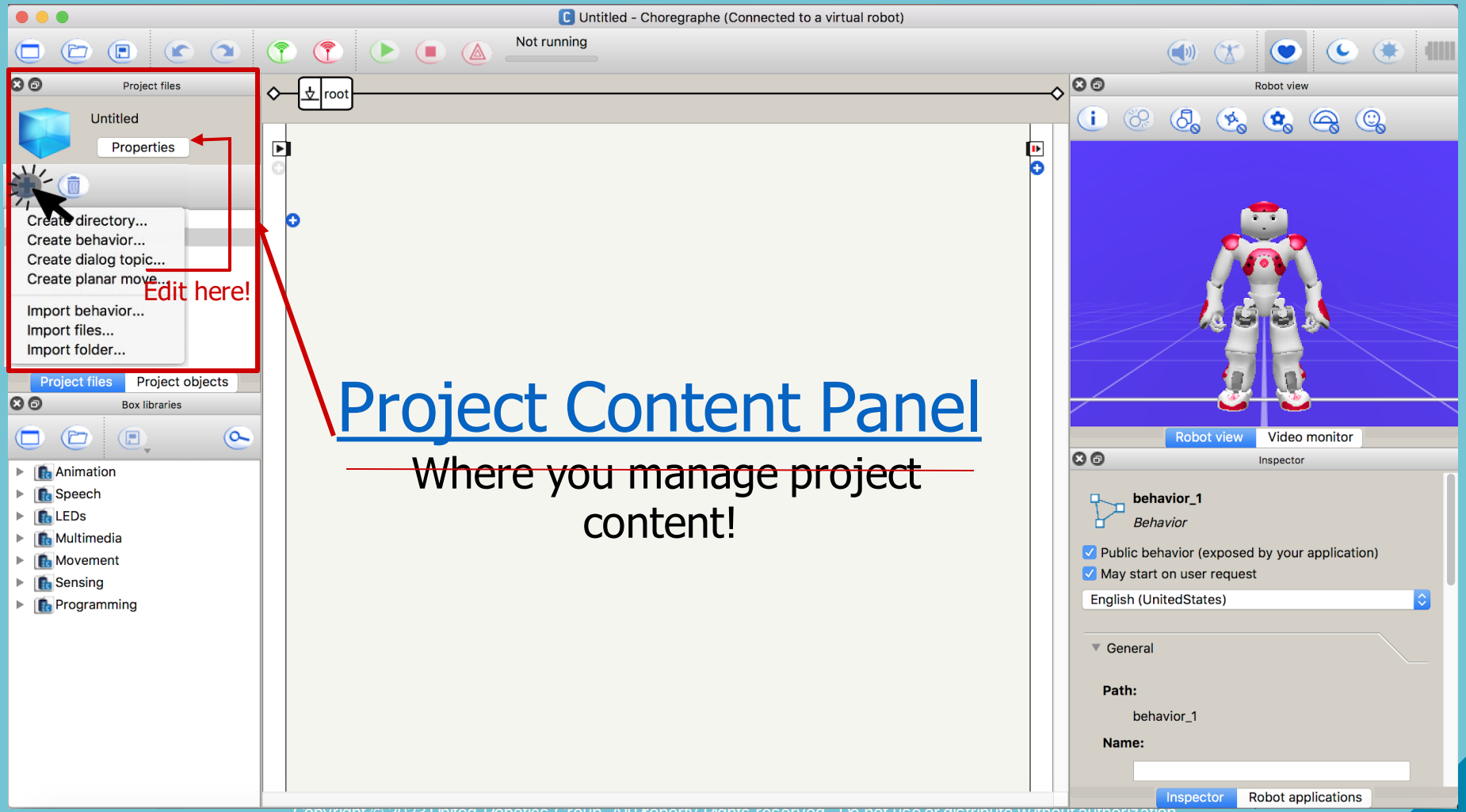
Choregraphe - Topic I

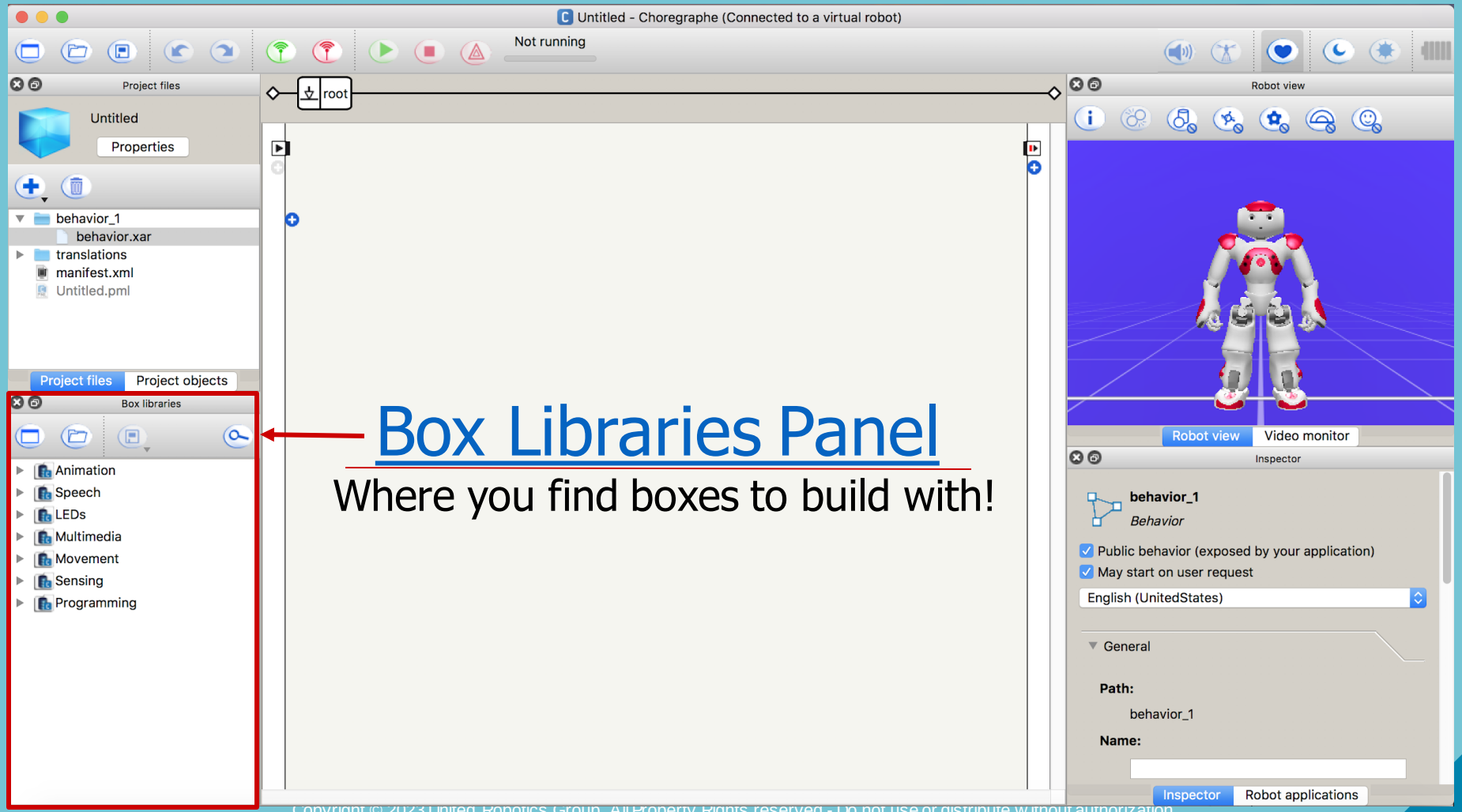
Overview

Tool introduction / Choregraphe



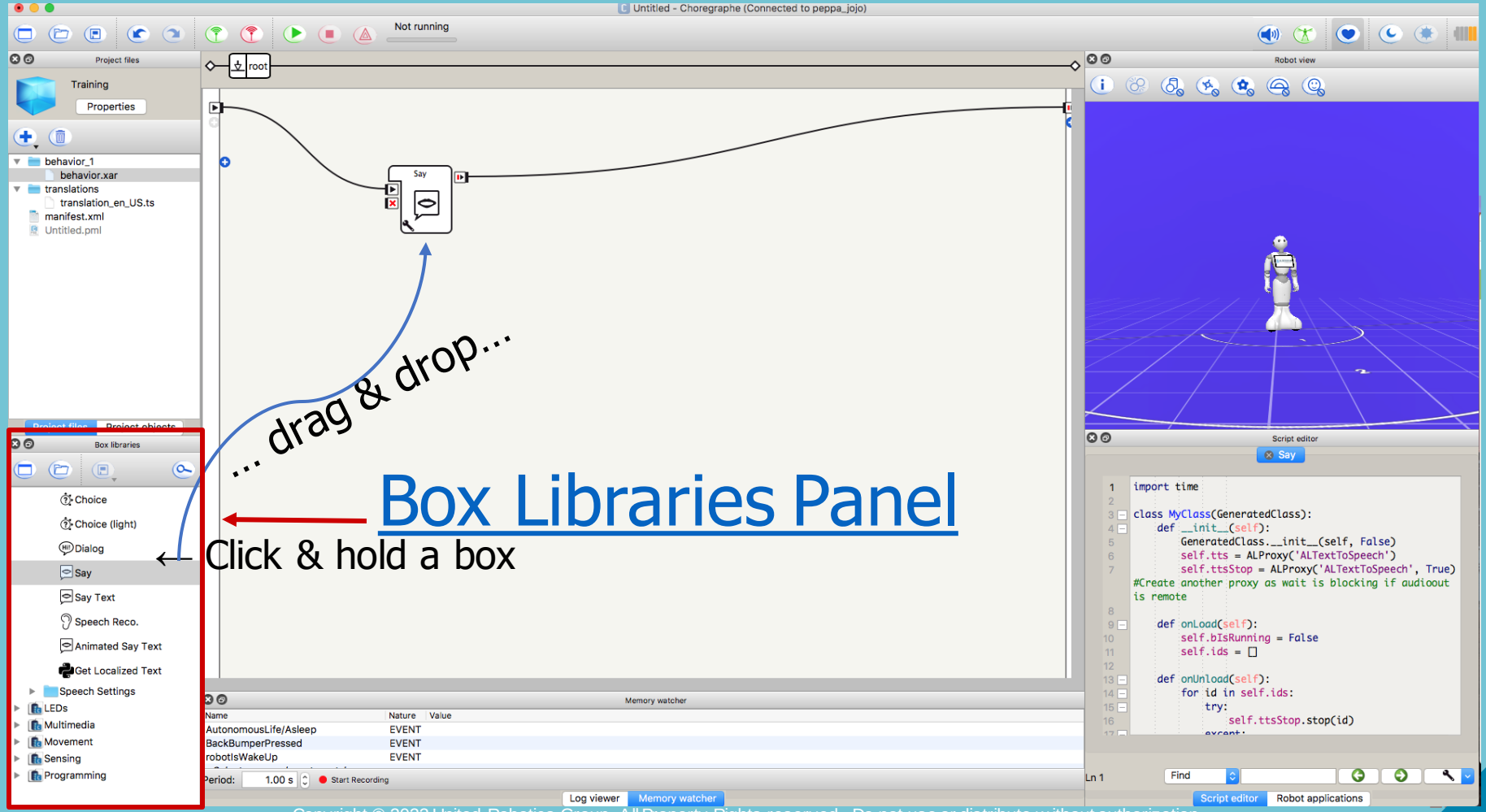


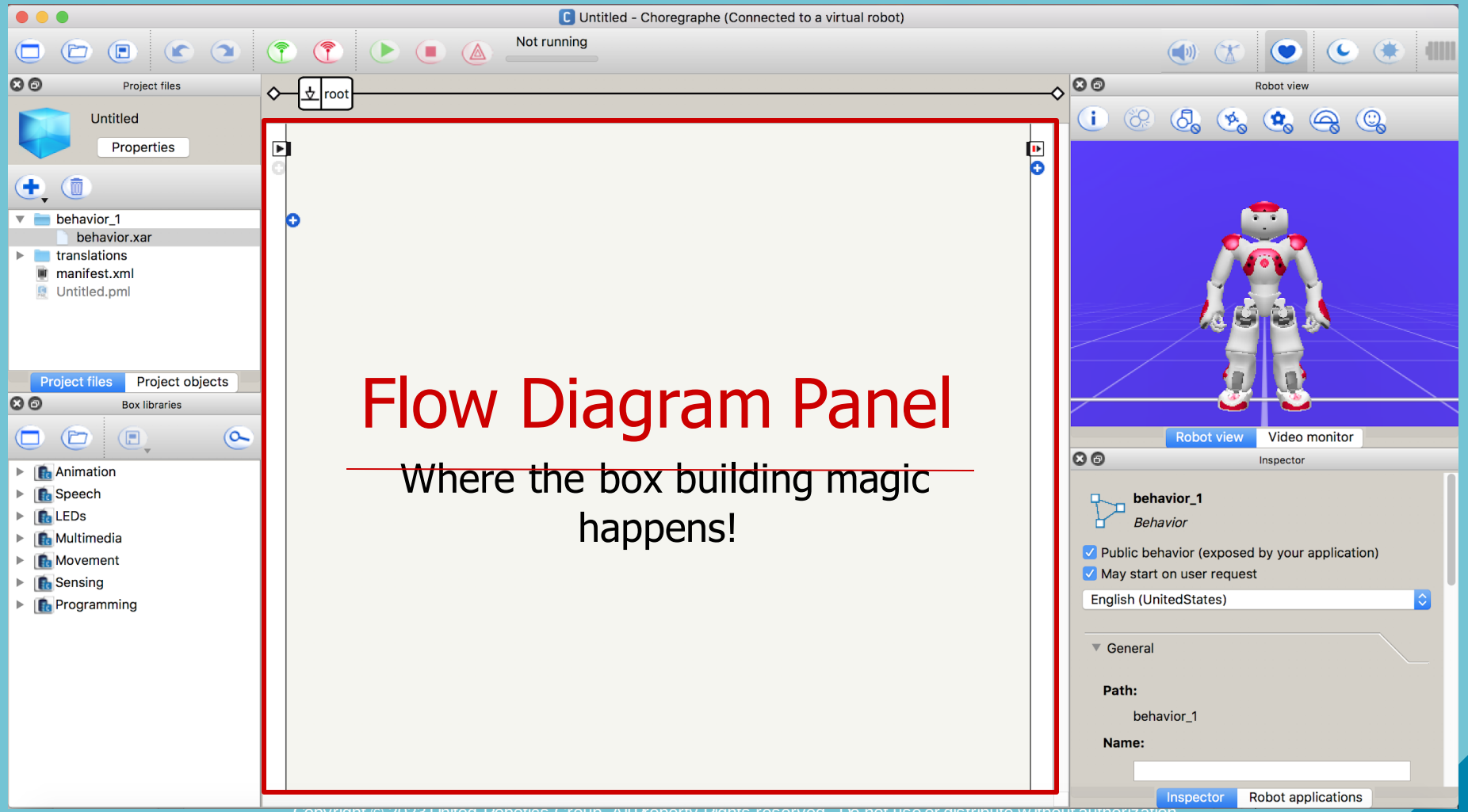


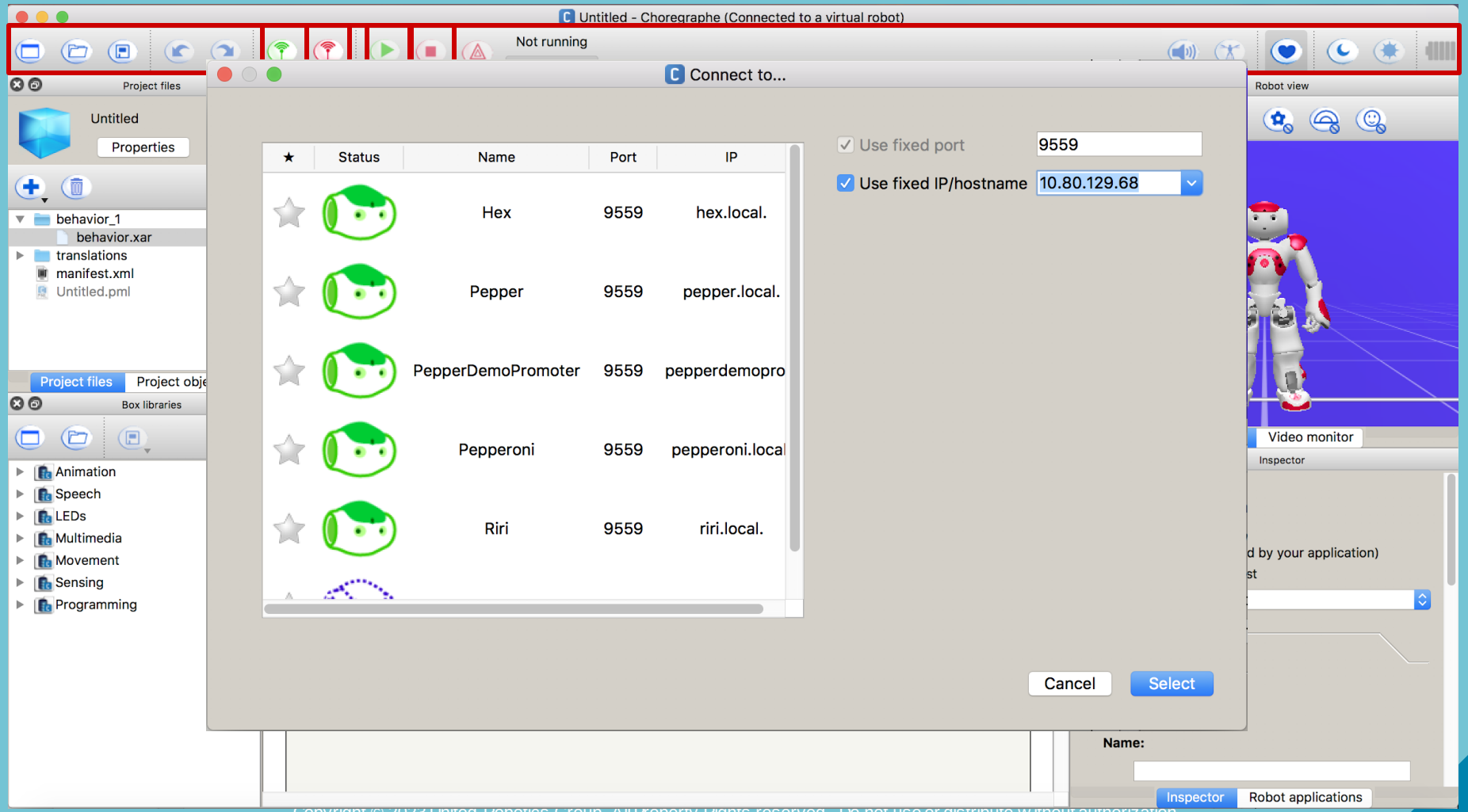


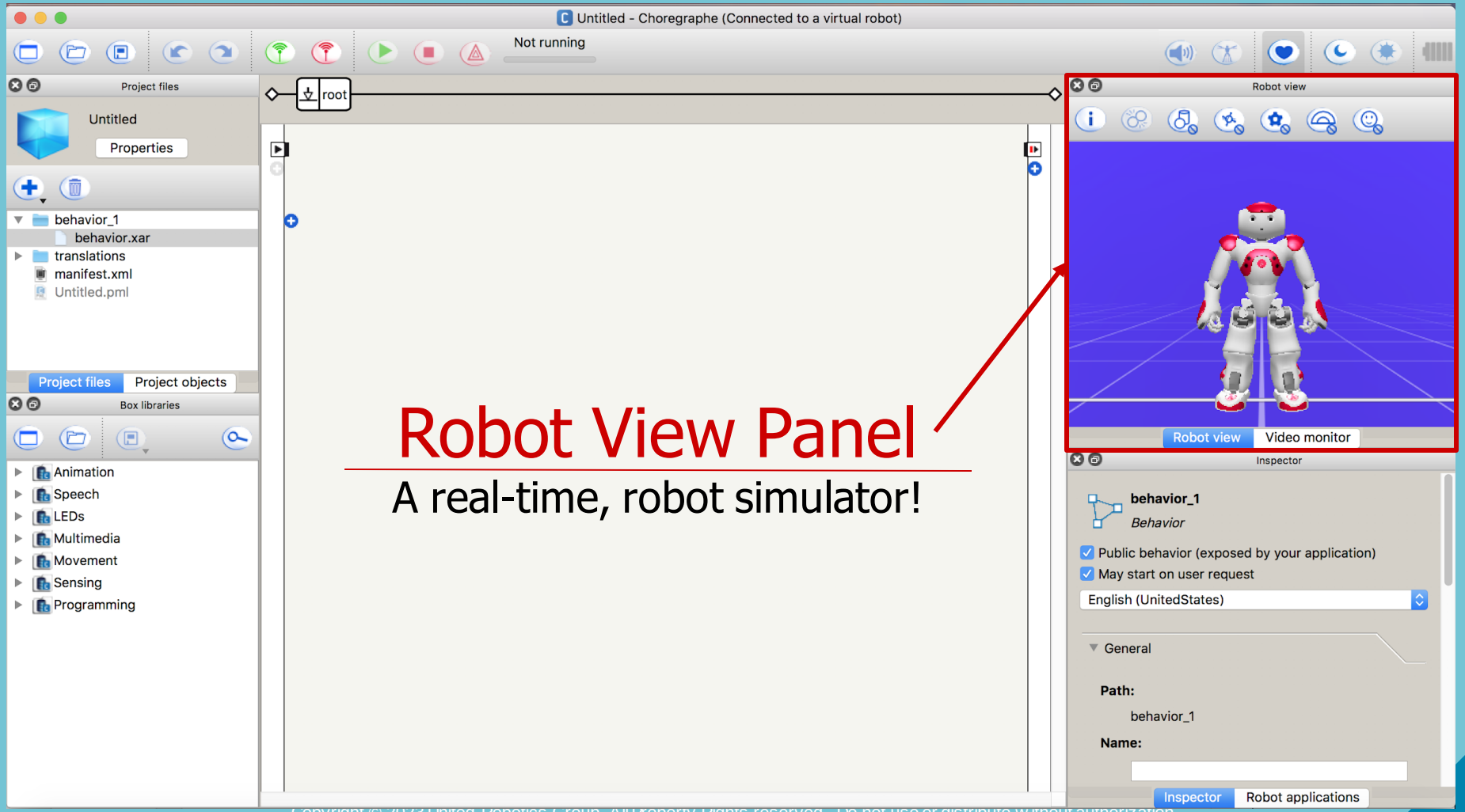
Box Libraries Panel

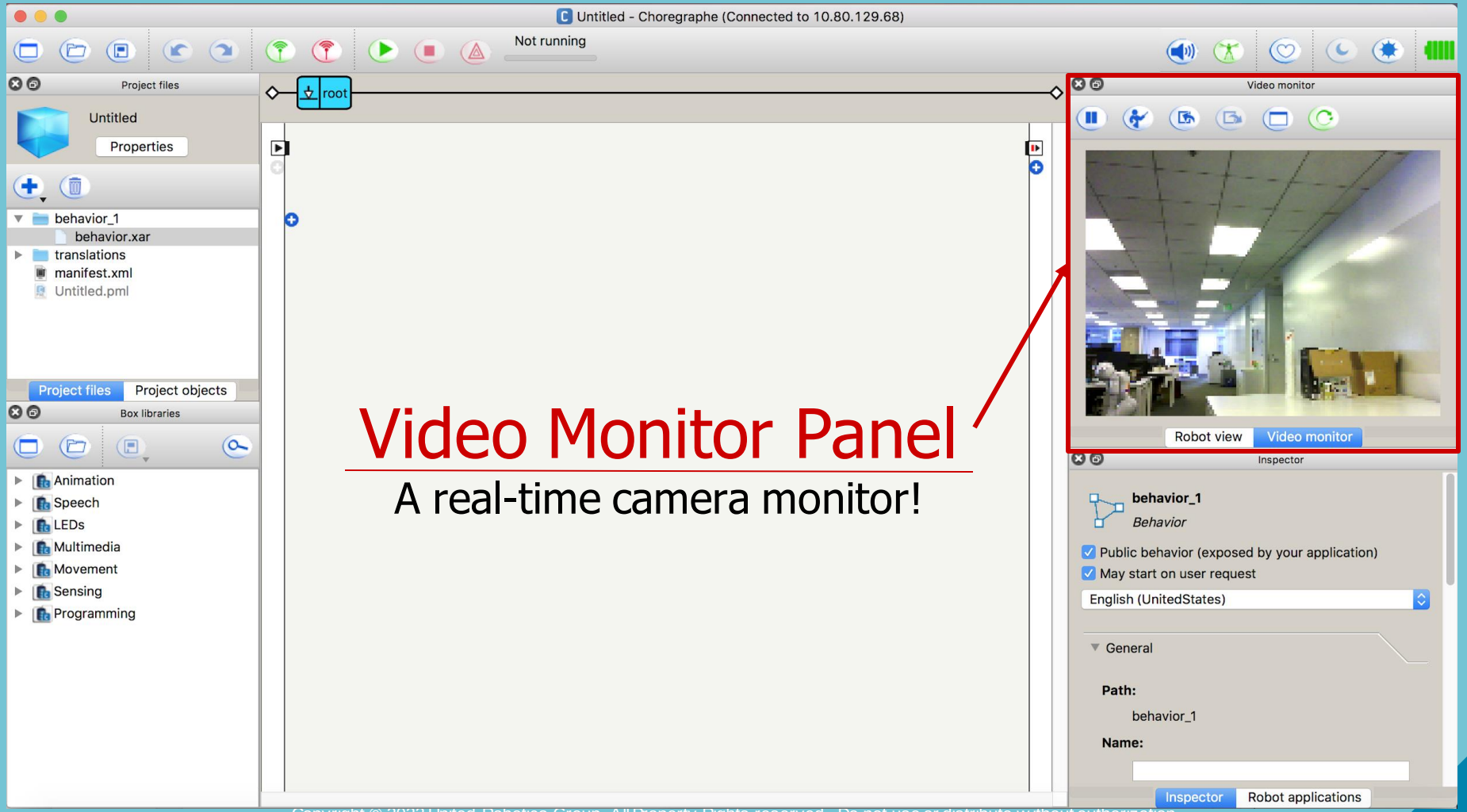
Where you find boxes to build with!





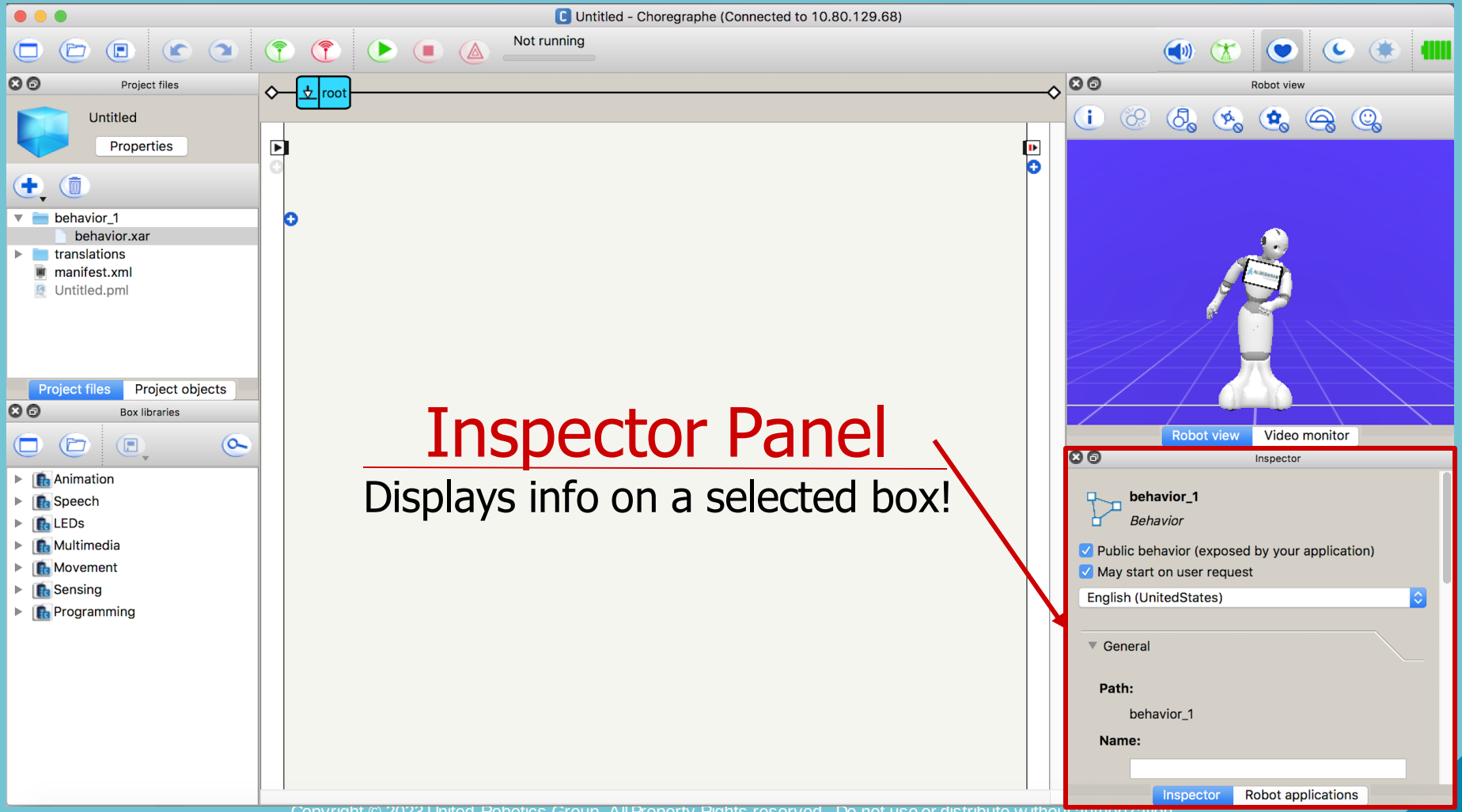






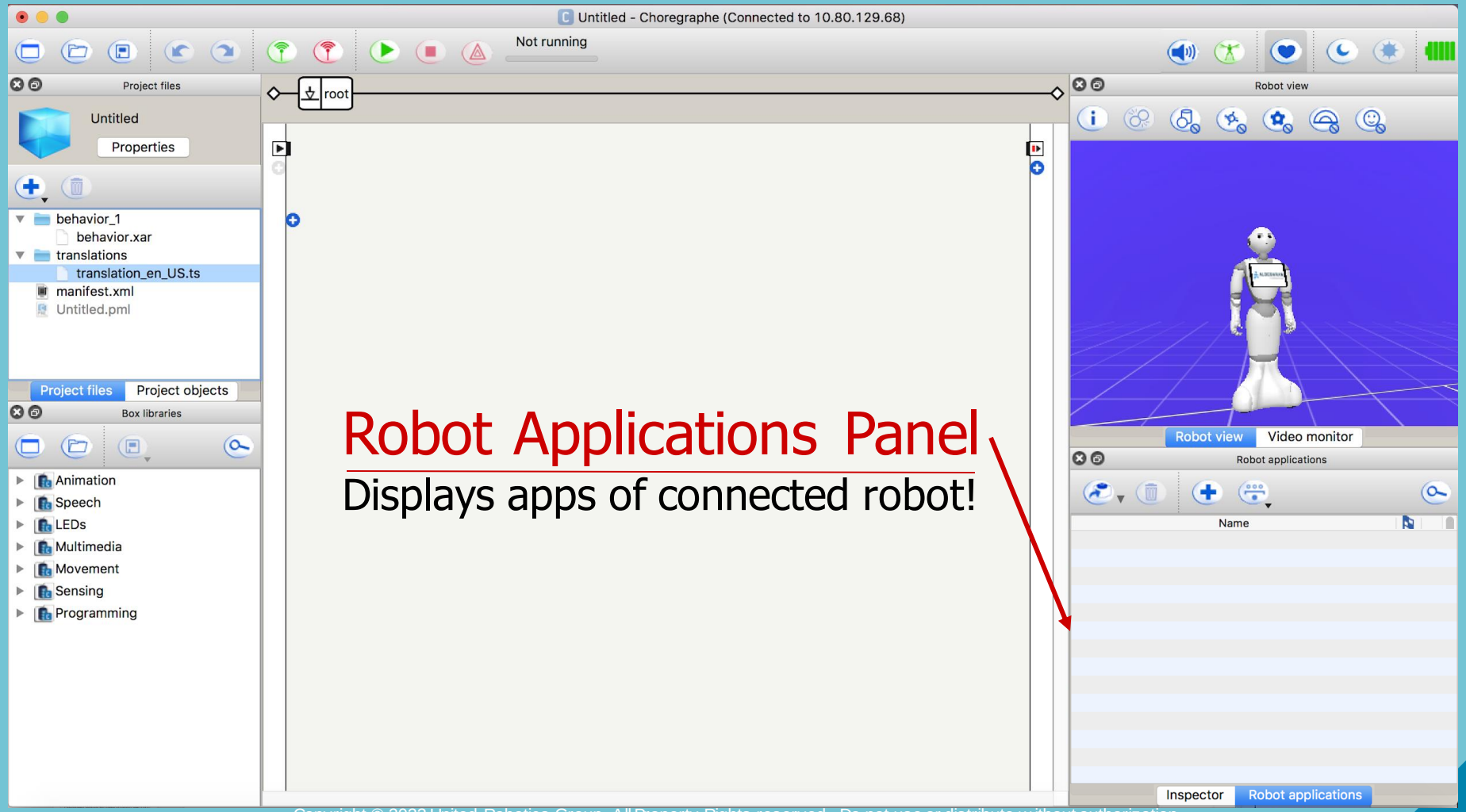
Video Monitor Panel

A real-time camera monitor!



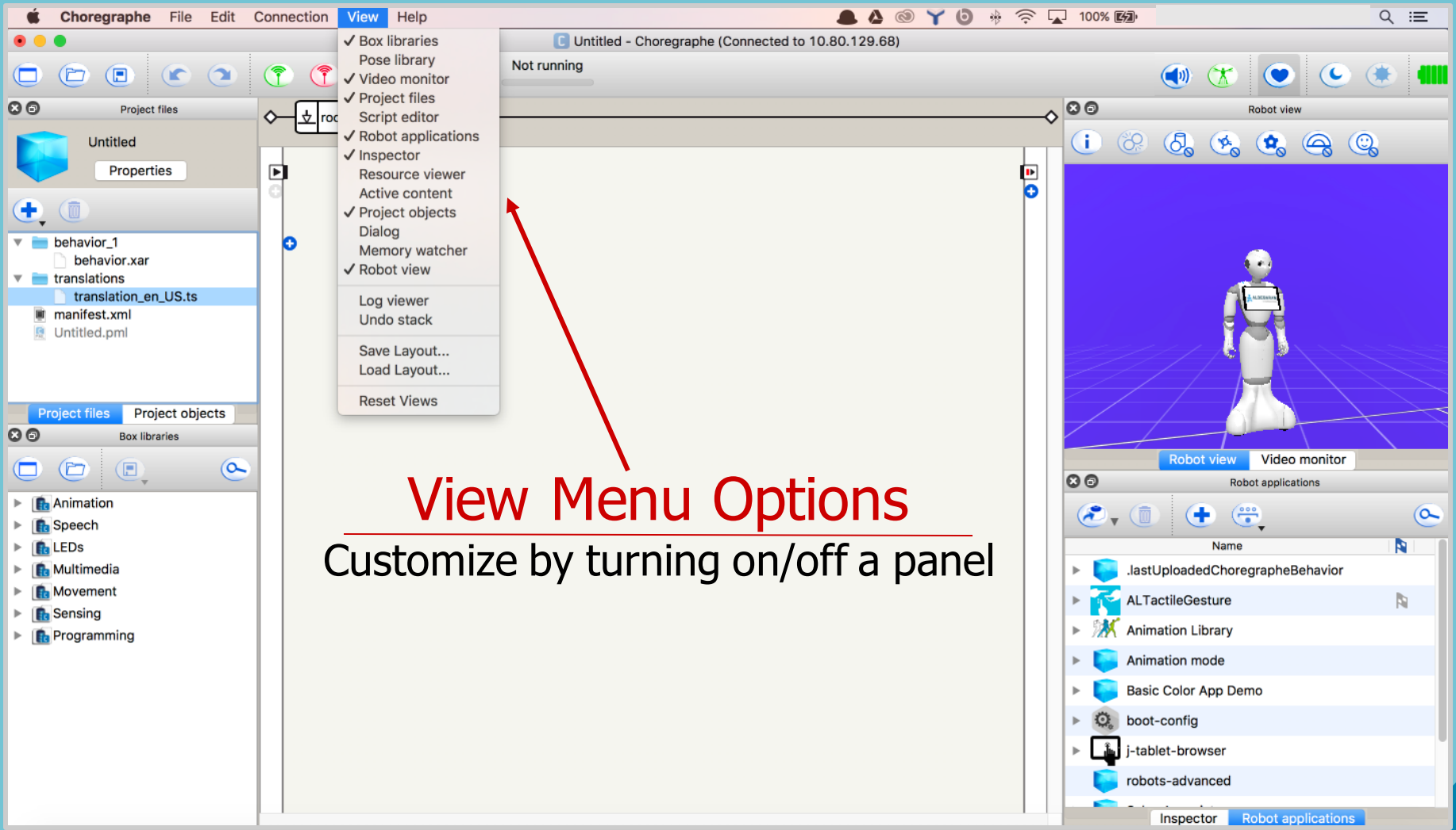
Inspector Panel

Displays info on a selected box!



Robot Applications Panel

Displays apps of connected robot!



View Menu Options

Customize by turning on/off a panel

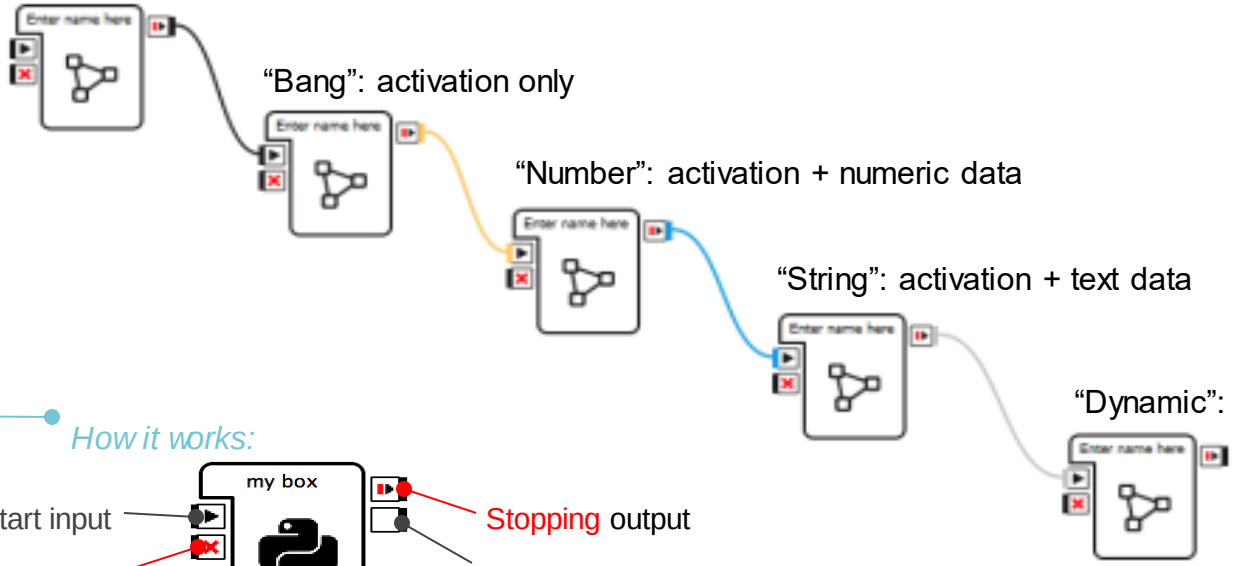
A Choregraphe project contains:

- A .pml file
- A manifest
- Any number of behaviors
- Any number of dialogs
- Resource files (text, music, videos, ...)
- Scripts and libraries

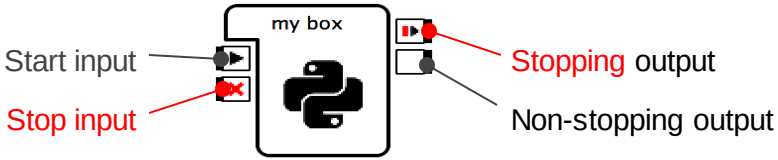
Choregraphe - Topic II

Boxes

Boxes: the building blocks of a Choregraphe app

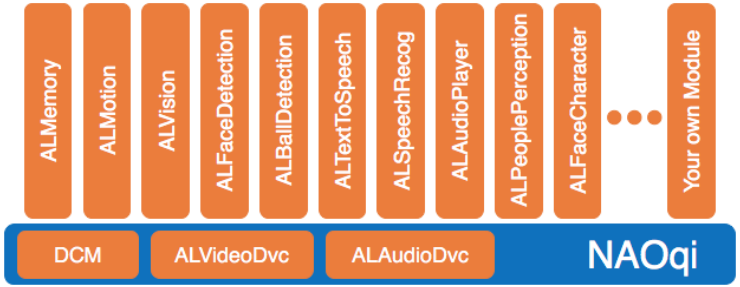


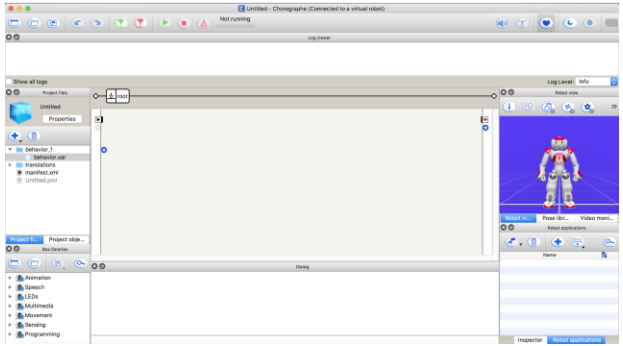
How it works:



GUI abstraction | *Choregraphe*

- The O.S. inside Pepper's Head is called ***NAOqi***



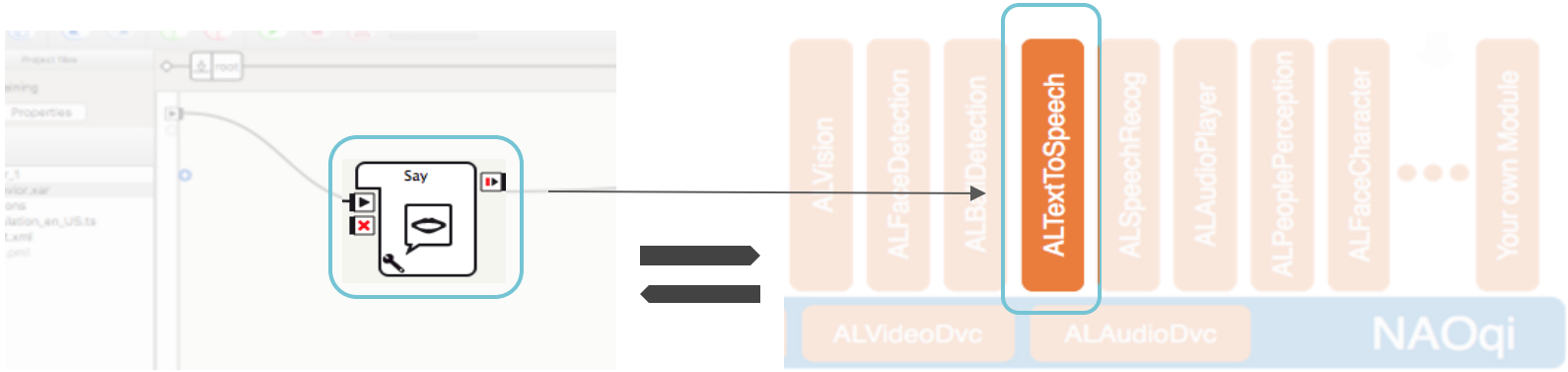


- ***NAOqi*** works with services

- ***Choregraphe*** abstracts the calling of services with a GUI

Boxes = GUI abstraction | *Choregraphe*

- Choregraphe abstracts the calling of services with a GUI
- **Say** box => calls ***ALTextToSpeech*** service

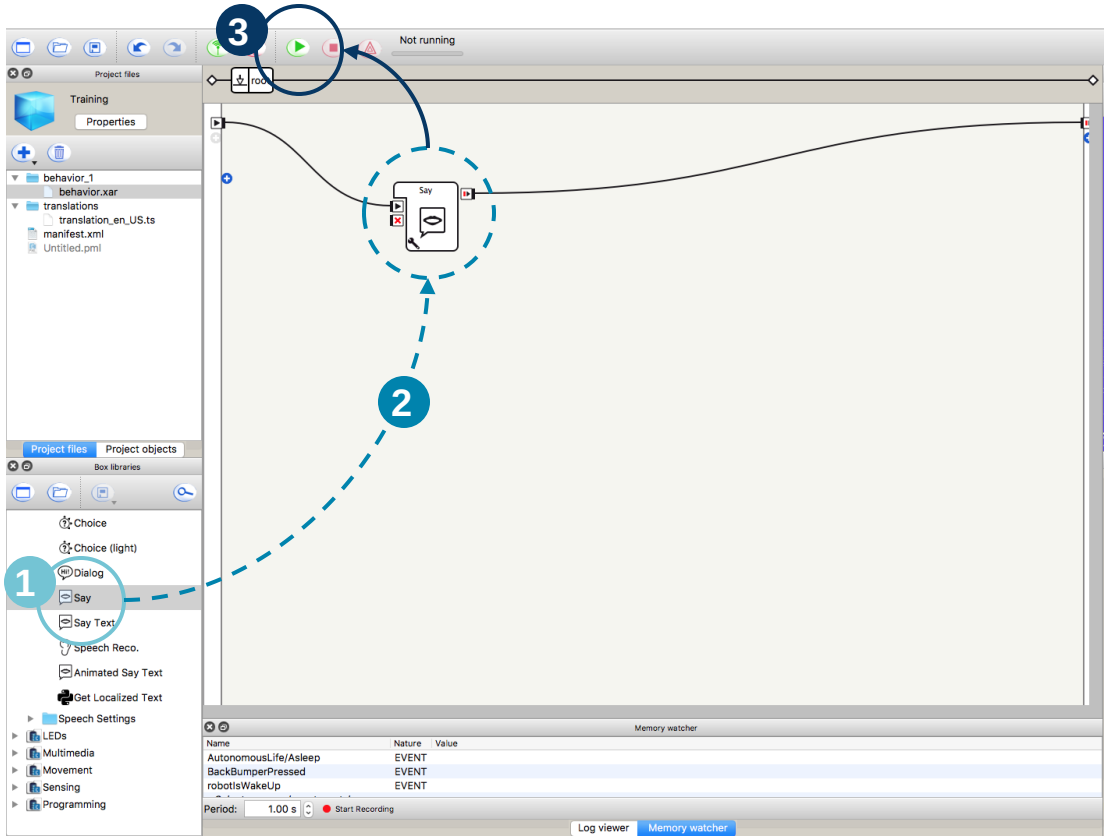


Building your first App | *Choregraphe*

Your First App!

> “Hello World!”

1. Click & hold “Say” Box
2. Drag & Drop
3. Then click play!



Type of Boxes | *Choregraphe*

Animation
- Timeline

Speech
- Dialog
- Say

Vision
- Face Detection

Multimedia
- Play Sound
- Play Video
- Record Sound

Touch
- Bumpers
- Tactile Head

LEDs
- Eye LEDs

Time
- Delay

Templates
- Python Script
- Diagram

Choregraphe - Topic IV

QiChat

Create your own Dialog | Choregraphe

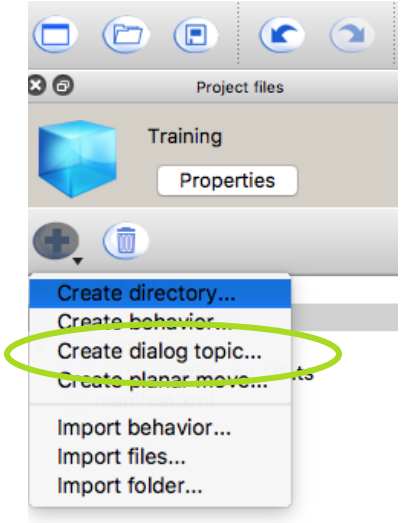
Now, discuss with your robot…

1. Click + then “Create dialog topic…”
2. Give a topic name (/!\ no space), select English and click OK
3. Locate the `.dlg` file
4. Drag & drop it into the program to create a box; link it
5. Edit the `.top` file…
6. Add as many rules as you want at the end of the file:

```
 u:( hello ) Hello sir!
u:( how are you ) I'm good, thanks!
```

[*Syntax:*] → u: *(what **user** says as prompt) what **robot** says in response*

7. Play the behavior!



More…

Lists:

[One choice is required]

```
u:( good [morning day afternoon]) …

u:( [ "good [morning day afternoon]" hello hi] ) …

u:( [ "good [morning day afternoon]" hello hi]
                                    {pepper you all everyone} ) …
```

{Optional choice}

A little more on dialog…

- u:(what **user** says) what **robot** says
- u:(**e:event**) what robot says
- u:(…) **$output_of_the_box=123**
- u:(…) **$memory_event=123**
- u:(…) **^run**(*application_id* / *behavior_id*)

Recommended architecture:

concept:(hello) [hello hi hey]

concept:(good) {pretty very} [well good]

- Separate the content (concepts) from the logic
- Put concepts at the top of the file for readability

u:(~hello) hi Jonas !
^gotoReactivate(**askhowareyou**)

proposal: **%askhowareyou** How are you?
u1:(bad) oh no! **^gotoReactivate**(...)
u1:(~good) I'm happy to hear that!

- Use "proposal blocks" & "tags" to jump to different dialog sections

Tags | Choregraphe

```
u:(~hello) hi Jonas ! ^gotoReactivate(askhowareyou)

proposal: %askhowareyou How are you?
u1:(bad) oh no! ^gotoReactivate(...)
u1:(~good) I'm happy to hear that!
```

The bolded text above, %**askhowareyou**, is a tag:

- You can use any alphabetic name ***(/!\ NO spaces!)***
- A tag is active by default
- You activate or deactivate a proposal block using tags
- A tag can be used any number of times
- When you activate a tagged proposal, it is read by the robot

Choregraphe - Topic V

Animation

Animation Library | *Animate The Robot*

In the animation library, there are 200+ movements available.

Use them in a dialog:

^run	-	starts and blocks until the movement is finished
^start	-	starts and continues while the movement is playing
^wait	-	blocks until the movement is finished (use it after a ^start)
^stop	-	stops a movement

```
u:(hello) ^run(animations/Stand/Gestures/Hey_1) Hey There!
```

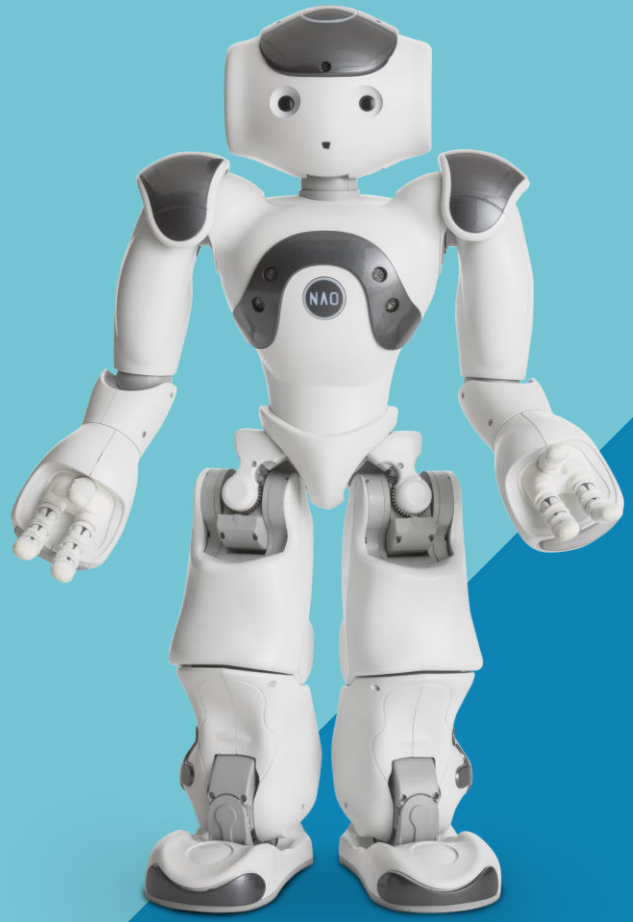
Create a movement | *Animate The Robot*

If you need more animations, you need to create them!

1. Create a new box “timeline”
2.  Disable autonomous life
3.  Wake up the robot
4.  Activate “animation mode”
5. Touch the hand to move the arm; store the position by tapping the head

Python

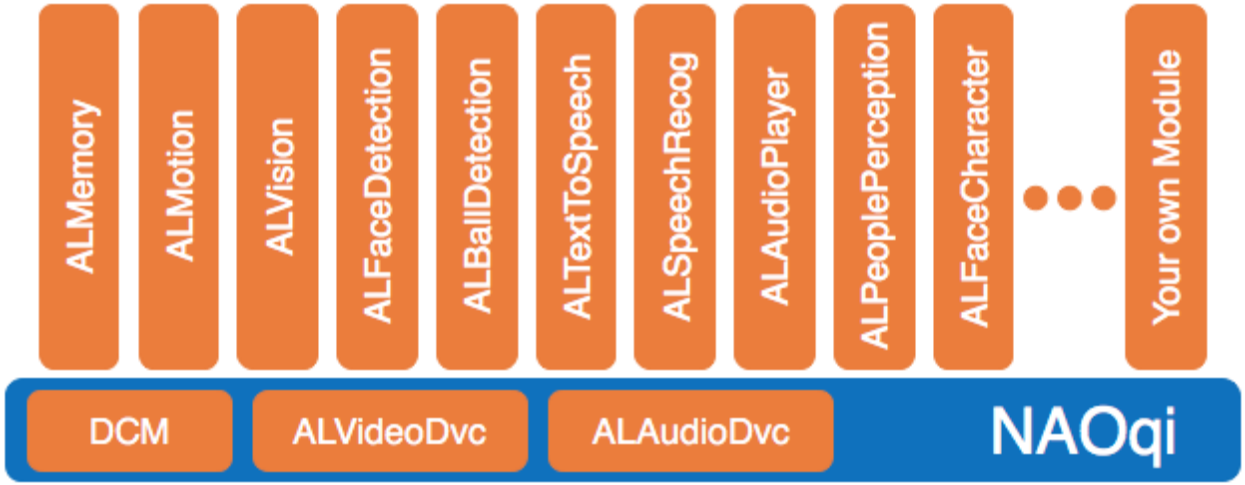
Use the full NAOqi SDK



Python - Topic I

The SDK

- NAOqi works with services



http://doc.aldebaran.com/2-8/naoqi/

Some useful APIs:

- ALTextToSpeech.say("hello everybody")
- ALAnimatedSpeech.say("hello everybody")
- ALMotion (everything movement related - many methods)
- ALLeds (many methods)
- ALBehaviorManager

All covered on http://doc.aldebaran.com/2-8/naoqi/

NAOqi | Pepper | NAO | Romeo

NAOqi - Developer guide
- Getting Started
- Creating an application
- Programming for a living robot
- Other tutorials
- Choregraphe Suite
- SDKs
- **NAOqi APIs**
  - NAOqi Core
  - NAOqi Emotion
  - NAOqi Interaction engines
  - NAOqi Motion
  - NAOqi Audio
  - NAOqi Vision
  - NAOqi People Perception
  - NAOqi Sensors & LEDs
  - LoLA
  - Types
- qi Framework
- Former NAOqi Framework

NAOqi APIs

On this page

Core

Read also the NAOqi Core introduction.

- ALBehaviorManager API | overview
- ALConnectionManager API | overview
- ALDiagnosis API | overview
- ALExpressionWatcher API | overview
- ALExtractor API | overview
- ALKnowledge API | overview
- ALMemory API | overview

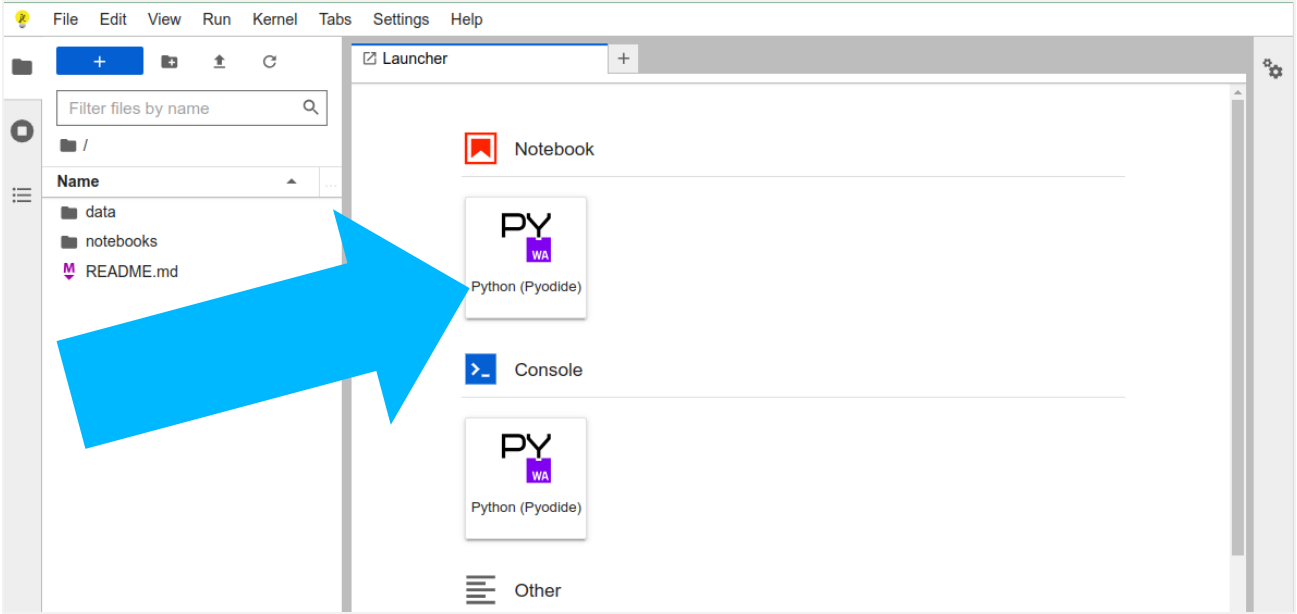
Python - Topic II

Jupyter Notebook

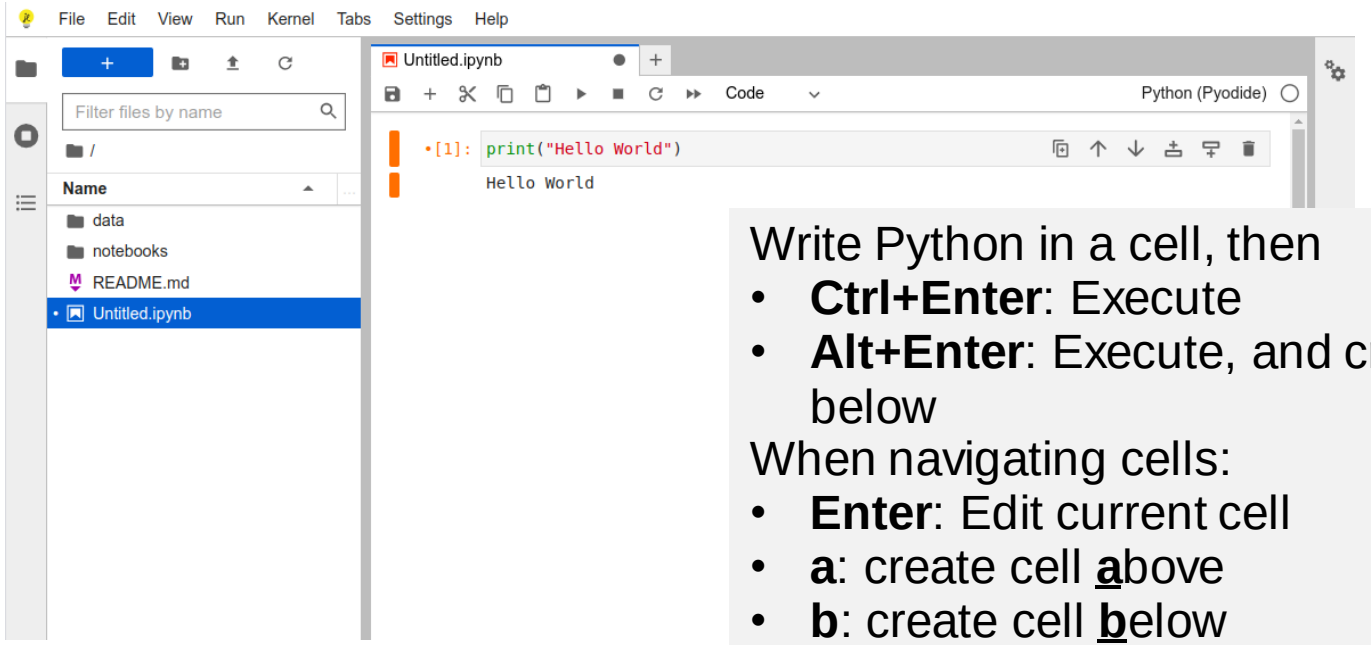
Connecting to Jupyter Lite

1) Connect to **`http://<NAO's IP>/apps/jupylite-nao`**

2) Create a new notebook



Notebook basics



Write Python in a cell, then

- **Ctrl+Enter**: Execute
- **Alt+Enter**: Execute, and create new cell below

When navigating cells:

- **Enter**: Edit current cell
- **a**: create cell **a**bove
- **b**: create cell **b**elow
- **d** twice: **d**elete cell

Connecting to NAO with ipynao

[2]:
```
%pip install ipywidgets==8.0.7
%pip install ipynao==0.7.3
import ipynao
nao = ipynao.Robot()
nao.connect()
nao
```

[2]:
```
Connected
Task completed
```

[3]:
```
nao.ALTextToSpeech.say("hello world")
```

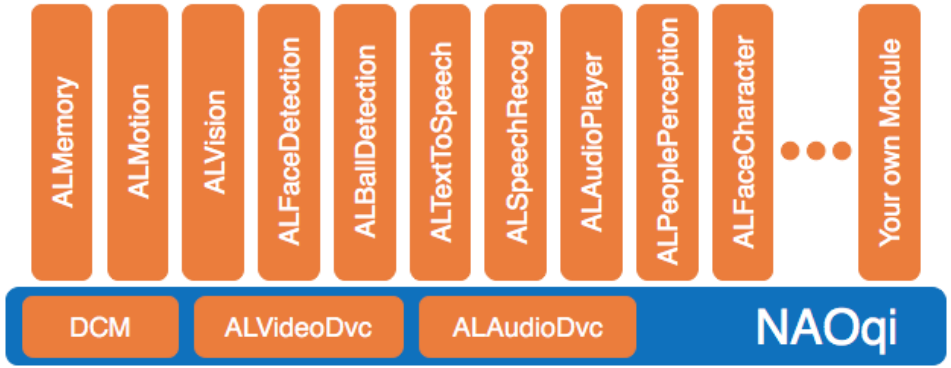
[3]:
```
<Task pending name='Task-66' coro=<NaoRobotService.call_serv
ice() running at /lib/python3.11/site-packages/ipynao/nao_ro
bot.py:40> cb=[WebLoop._decrement_in_progress()]>
```

What did we do?

```
[ ]: nao.ALTextToSpeech.say("hello world")
```

NAOqi service

Function call



A few more APIs to try

Animated speech:

```
[ ]: nao.ALAnimatedSpeech.say("hello world")
```

LEDs:

```
[ ]: nao.ALLeds.rasta(2.0) # duration
```

More on Animated Speech

Contrast these two:

```
[ ]: nao.ALTextToSpeech.say("hello everybody")
```

```
[ ]: nao.ALAnimatedSpeech.say("hello everybody")
```

ALAnimatedSpeech also supports more syntax:

```
[ ]: nao.ALAnimatedSpeech.say("I can add a \\pau=1000\\ pause")
```

Animated Speech Syntax: voice

```
# Say the sentence with a pitch of +50%
"\\vct=150\\Hello my friends"
# Say the sentence 50% slower than normal speed
"\\rspd=50\\hello my friends"
# Say the sentence with a volume of 50%
"\\vol=50\\Hello my friends"
# Change the tone (available: normal, joyful, didactic)
"\\style=joyful\\ Today I am feeling happy."
# Reset with \\rst\\
"\\vct=150\\\\rspd=50\\Hello my friends.\\rst\\ How are you ?"
```

Documentation:

https://doc.aldebaran.com/2-8/naoqi/audio/altexttospeech-tuto.html

Animated Speech Syntax

Playing animations:

```
"^runTag(me) My name is Nao."

"^startTag(hello) Hello. ^waitTag(hello)"

"^startTag(hello) Hello there. ^stopTag(hello) My name is Nao."
```

Documentation:

http://doc.aldebaran.com/2-8/naoqi/audio/alanimatedspeech.html

Animated Speech Syntax

Playing animations and sounds:

```
"Hello! ^runTag(me) My name is Nao.

^start(animations/Sit/Emotions/Positive/Happy_1) Nice to meet you
^wait(animations/Sit/Emotions/Positive/Happy_1)

^startSound(soundsetaldebaran/enu_ono_wouah_crazy)"
```

Documentation:

http://doc.aldebaran.com/2-8/naoqi/audio/alanimatedspeech.html

Different syntaxes for calling the same API

In Jupyter:

- nao.ALTextToSpeech.say("hello")

In Choregraphe:

- self.session.service("ALTextToSpeech").say("hello")

or (deprecated):

- ALProxy("ALTextToSpeech").say("hello")

In command line:

- qicli call ALTextToSpeech.say hello

Create a movement | *Animate The Robot*

In Jupyter, getting a function call's response is tricky:

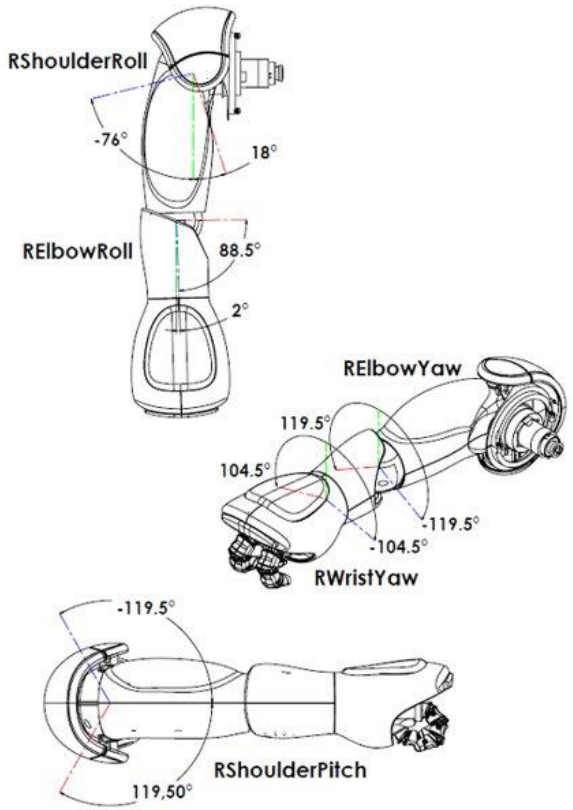
In one cell:

```
r = ALMotion.getAngles("LArm", True)
```

In the next cell

```
print(r.result)
```

Create a movement | *Animate The Robot*



Joint control API

ALMotion.getAngles(joints, useSensors)

- joints: "Body", "LArm", "RArm", "Head", "LElbowRoll", etc.
- useSensors: if False, use last command

ALMotion.setStiffnesses(joints, stiffnesses)

- Stiffnesses can be a list, or 0.0 (motors off), 1.0 (motors on), or in between

ALMotion.setAngles(joints, angles, fractionMaxSpeed)

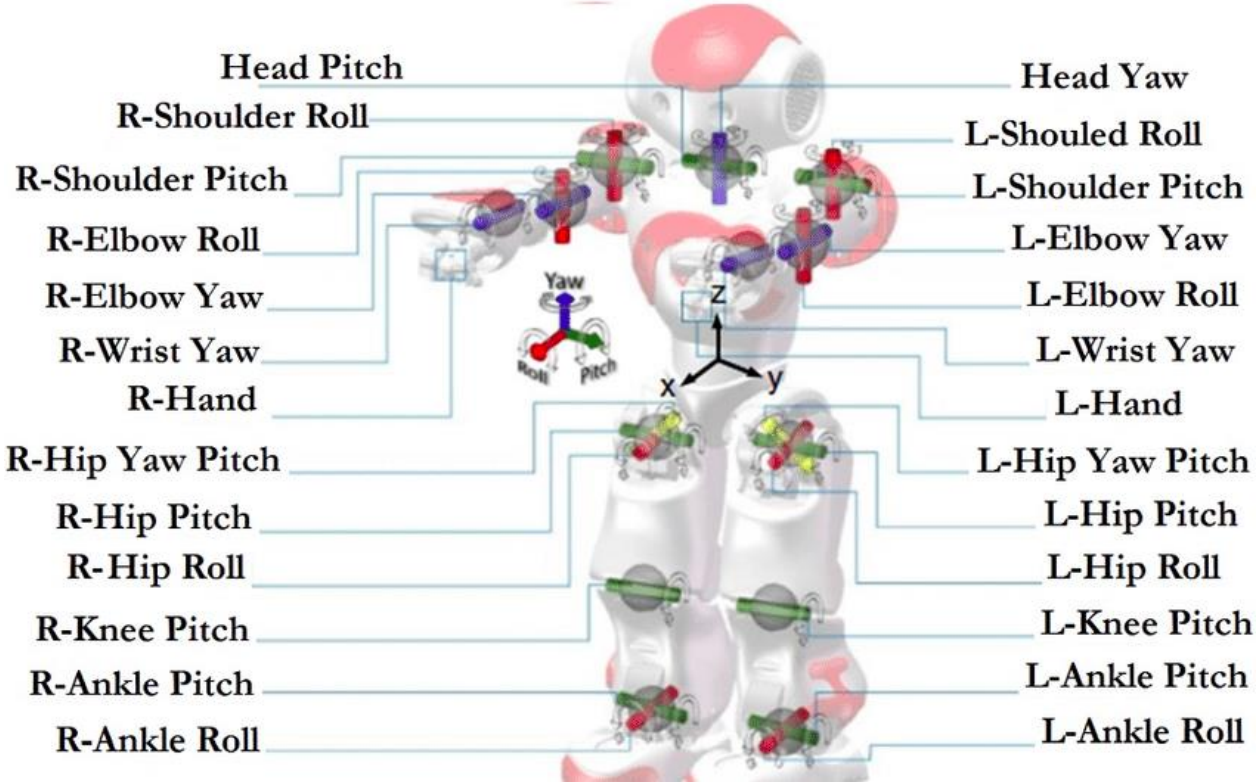
- Non-blocking call

ALMotion.angleInterpolationWithSpeed(joints, angles, fractionMaxSpeed)

- Blocking call (note: in Jupyter all calls are non-blocking)

Create a movement | *Animate The Robot*

Joint names



Create a movement | *Animate The Robot*

Make Nao **navigate**

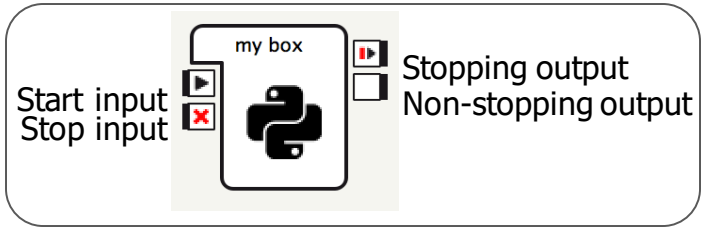
- **ALMotion.*moveTo***(x, y, theta) - straight, stops if encounters an obstacle
- **ALMotion.*moveToward***(x, y, theta)

Python - Topic III

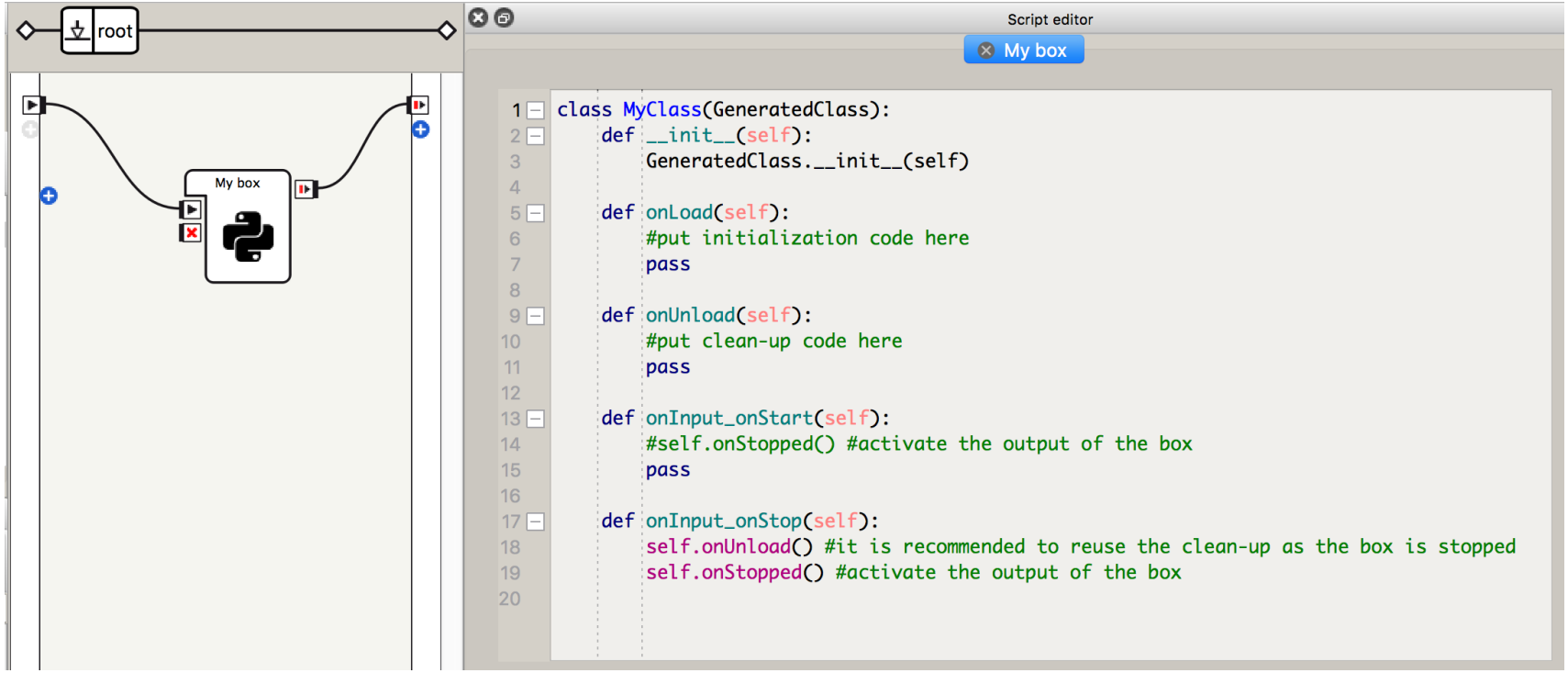
Python in Choregraphe

Now, let's create your own box…

1) Right click in the programming area
2) Add a new box:
   - Python!
3) Give it a name
4) Click OK
5) Double-click the box to access its code

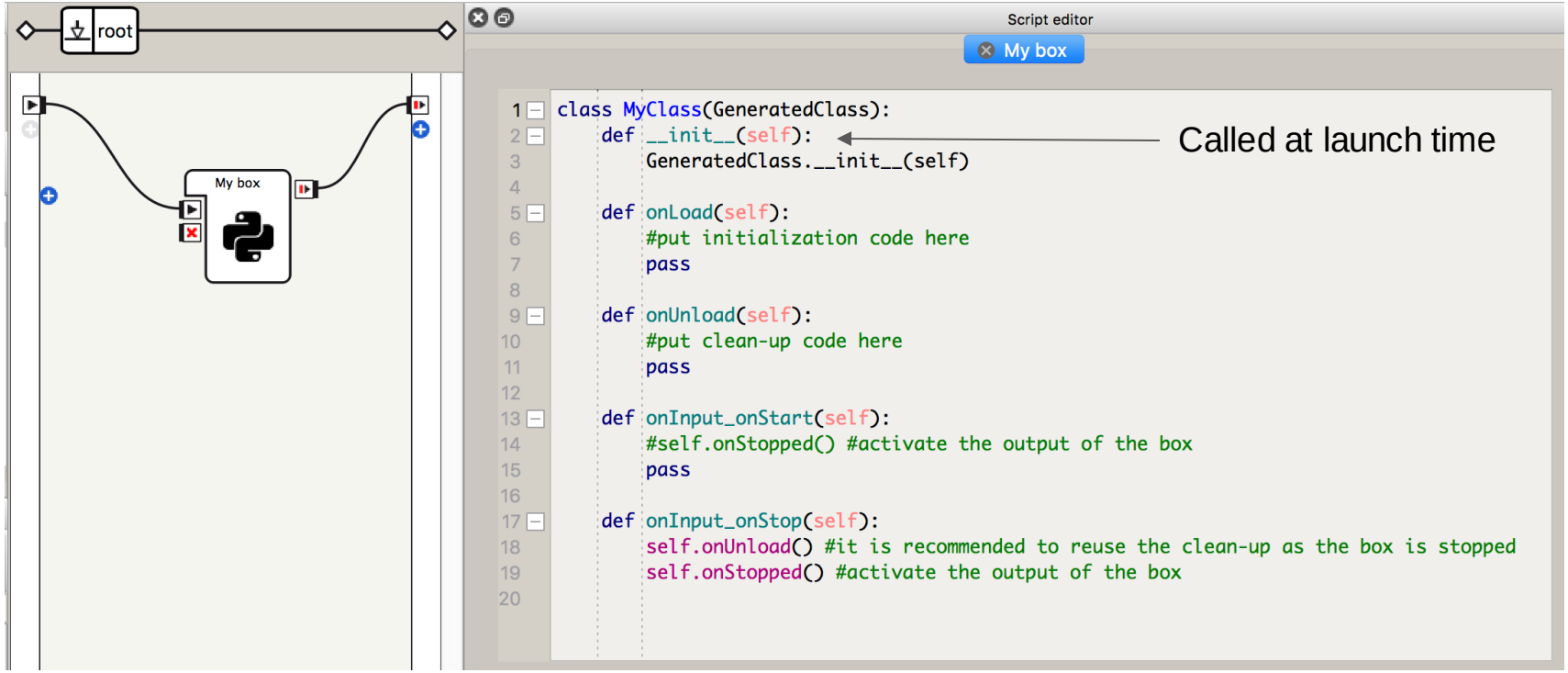


Python in Python Box Template | Choregraphe

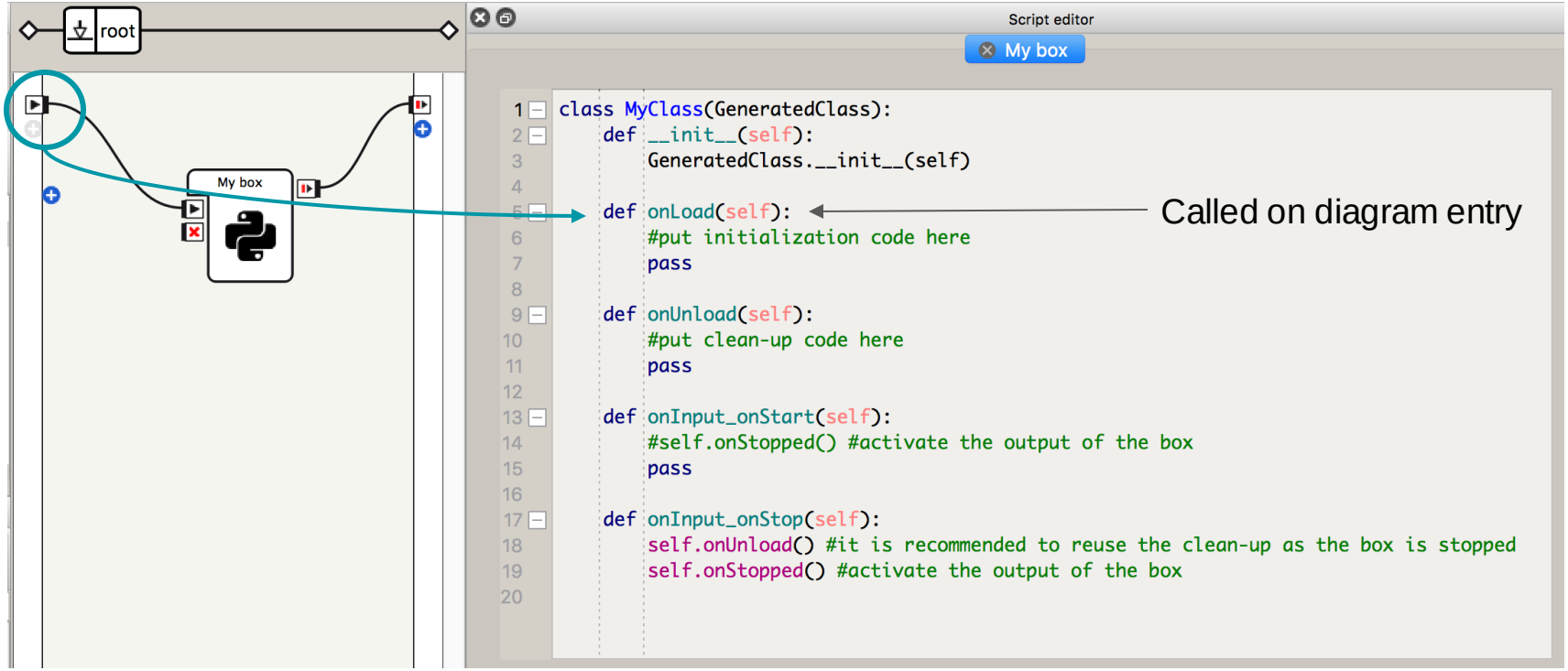



```
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)

    def onLoad(self):
        #put initialization code here
        pass

    def onUnload(self):
        #put clean-up code here
        pass

    def onInput_onStart(self):
        #self.onStopped() #activate the output of the box
        pass

    def onInput_onStop(self):
        self.onUnload() #it is recommended to reuse the clean-up as the box is stopped
        self.onStopped() #activate the output of the box
```

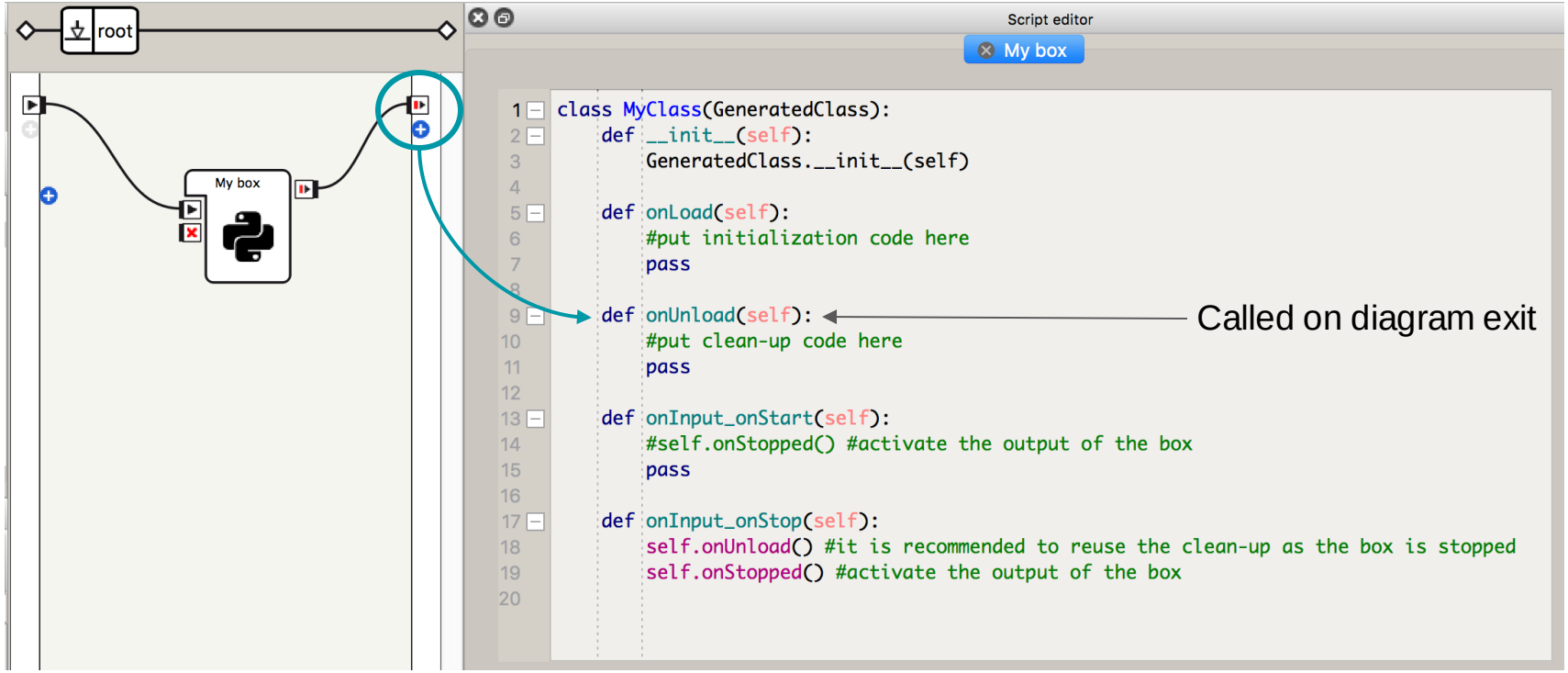
Launch = __init__()



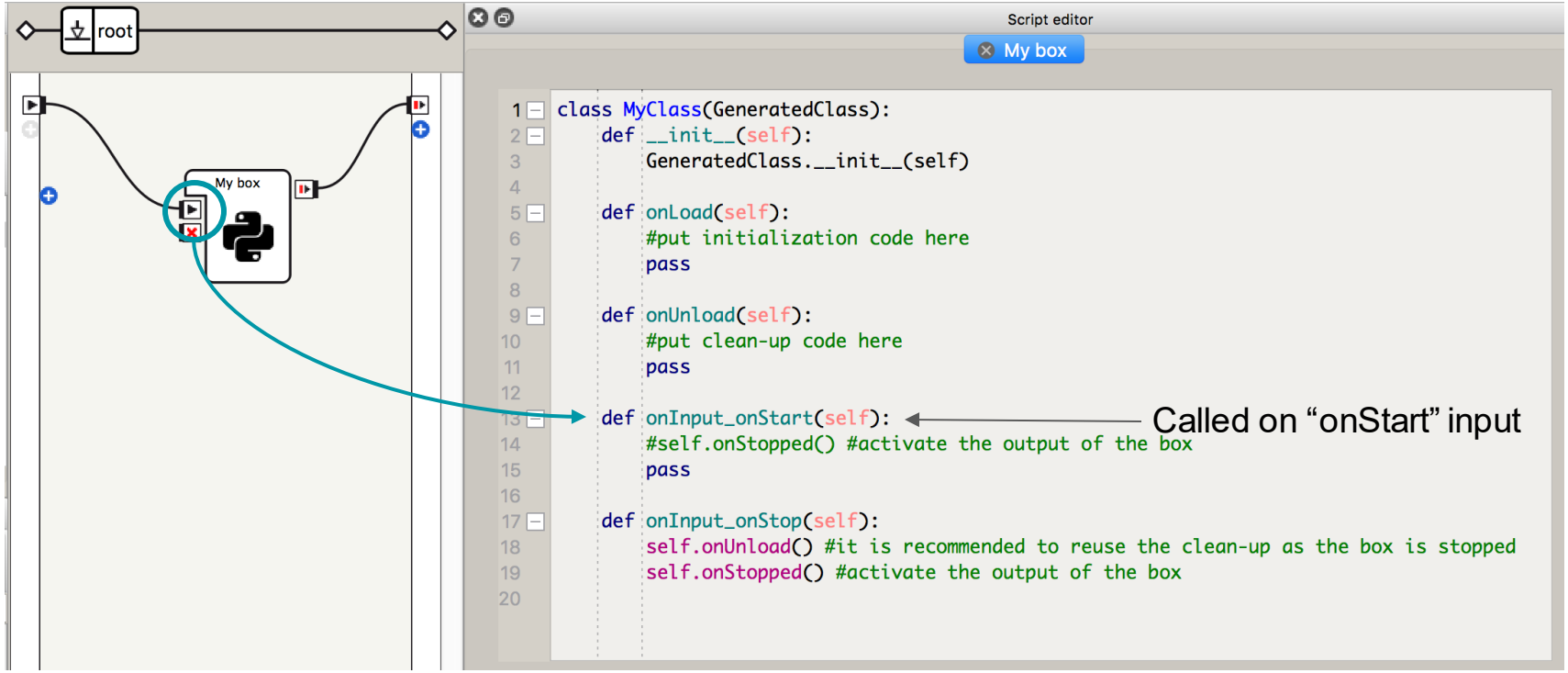
Python in Diagram Start = onLoad() | Choregraphe



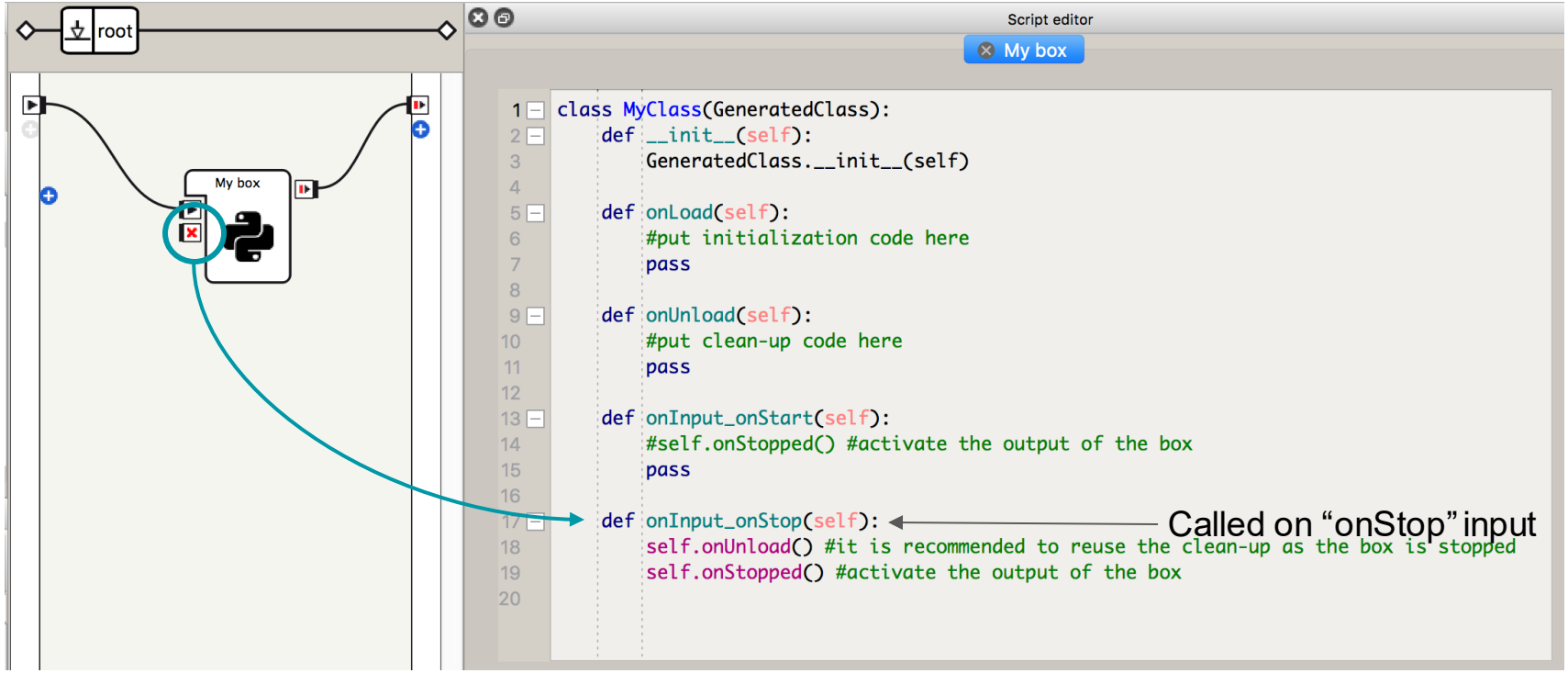
Python in Diagram Stop = onUnload() | Choregraphe



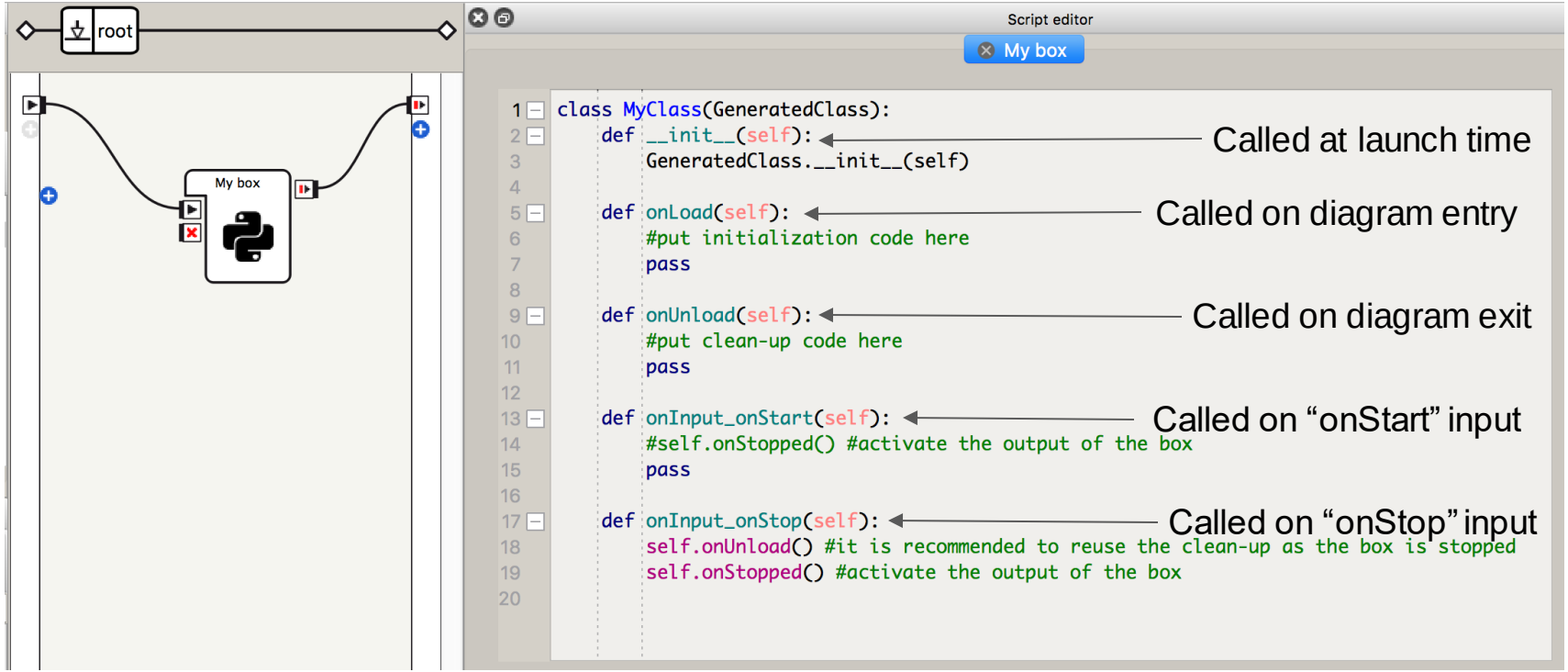
Python in Box Start = onInput_onStart() | Choregraphe

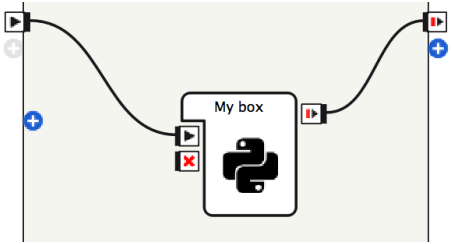


Python in Box Stop = onInput_onStop() | Choregraphe



Python in Python Box Template Recap | Choregraphe





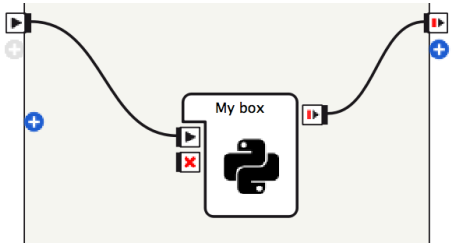
Python Boxes - Using the Logger

```
def onInput_onStart(self):
self.logger.info("Box is running!")
#~ self.onStopped() #~ activate output of the box
pass
```

See this log in the “log viewer” panel

Uncomment this to activate the output of the box

Python Boxes - Using Services




```
def onInput_onStart(self):
self.logger.info("Box is running!")

session = self.session()

tts = session.service("ALTextToSpeech")

tts.say("Hello from my box!")

self.onStopped() #~ activate output of the box
pass
```

Open a session to the robot
(`self.session()` is a helper provided by Choregraphe)

Access a service from the robot

Call one of the methods from that service

https://developer.softbankrobotics.com→ Nao6 → NAOqi developer guide → NAOqi APIs

Python - Topic IV

Standalone Python

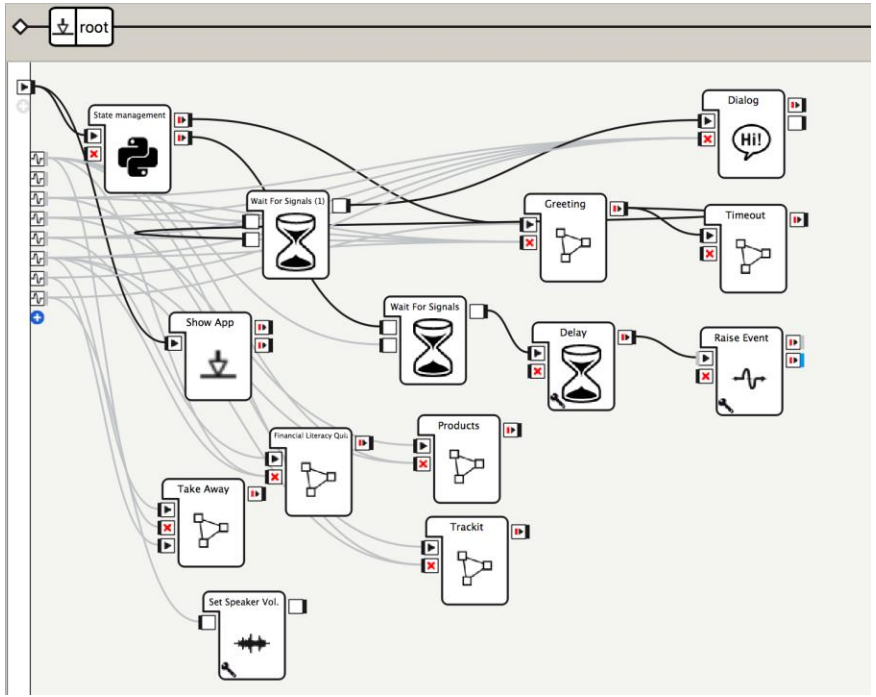
Production = Chaos! | Choregraphe | *Python*

A “production” app is usually 2000+ lines of code… this can easily reach 50+ boxes, and 1000+ wires...

As boxes, it is:

- HARD to write, share, version
- VERY HARD to organise, debug
- IMPOSSIBLE to understand, update

⇒ **We strongly advise to code in Python!**



Each box contains 15+ boxes.. They all run concurrently...

Standalone Python | *Python*

It's possible to create applications that are simply a Python script *outside* Choregraphe. This allows to:

- Organize your code in Python modules
- Import python libraries
- Create your own services
  maybe instead of ALAnimatedSpeech.say("hello") you can use your own ALSingingSpeech.say("hello")?

Standalone Python | *Python*

Example:
(using "stk"
"studio toolkit")

```python
import stk.runner
import stk.events
import stk.services
import stk.logging

class Activity(object):
    "A sample standalone app, that demonstrates simple Python usage"
    APP_ID = "com.aldebaran.demoapp"
    def __init__(self, qiapp):
        self.qiapp = qiapp
        self.events = stk.events.EventHelper(qiapp.session)
        self.s = stk.services.ServiceCache(qiapp.session)
        self.logger = stk.logging.get_logger(qiapp.session, self.APP_ID)

    def on_start(self):
        "Ask to be touched, waits, and exits."
        self.s.ALTextToSpeech.say("Touch my forehead.")
        self.logger.warning("Listening for touch...")
        while not self.events.wait_for("FrontTactilTouched"):
            pass
        self.s.ALAnimatedSpeech.say("Thank you!.")

if __name__ == "__main__":
    stk.runner.run_activity(Activity)
```

For doing this in Python 2:

- https://github.com/aldebaran/robot-jumpstarter/ (includes several templates)

In Python 3:

- https://github.com/EmileKroeger/mini_python3_nao_app

More documentation on both at:

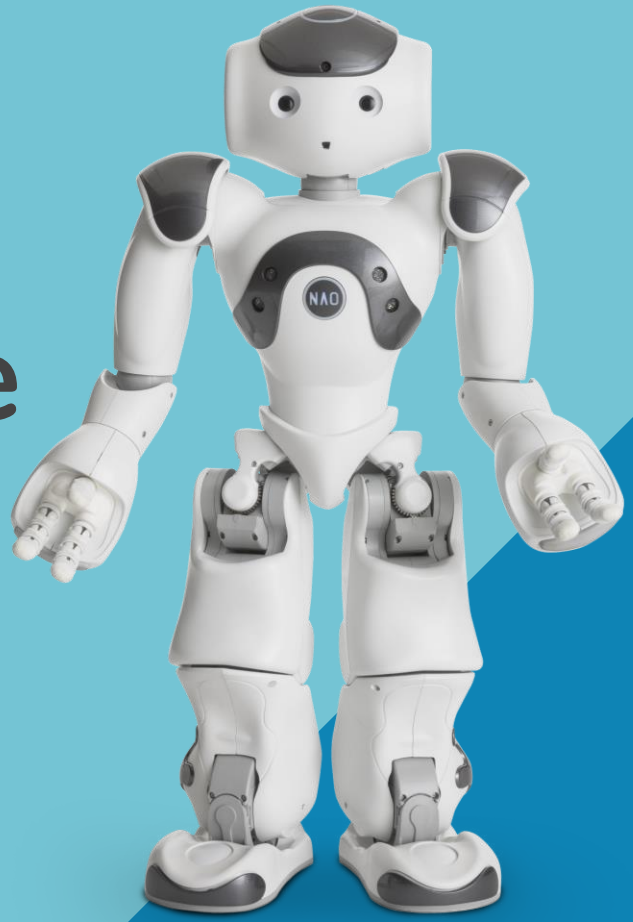
https://naosclassroom.github.io/

Congratulations!

You are ready to start prototyping

NAO Challenge

Let's look at physical tasks!

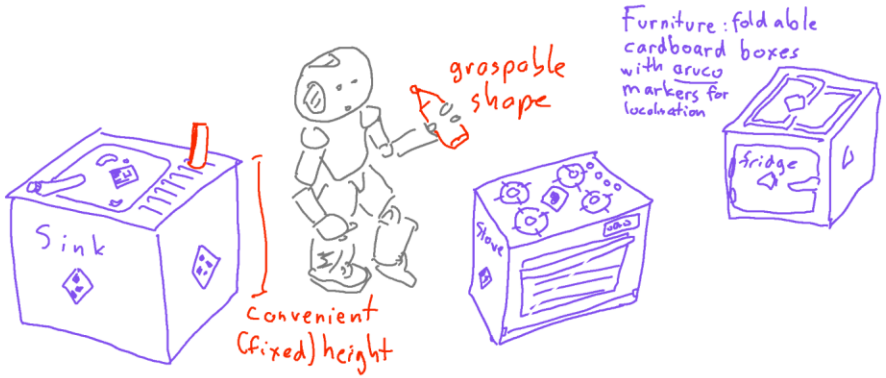


Physical tasks / *NAO Challenge*

Can use a pre-made "Aruco maze" package, which provides fuctions for

- Detecting markers
- Moving along markers

Physical tasks / *NAO Challenge*

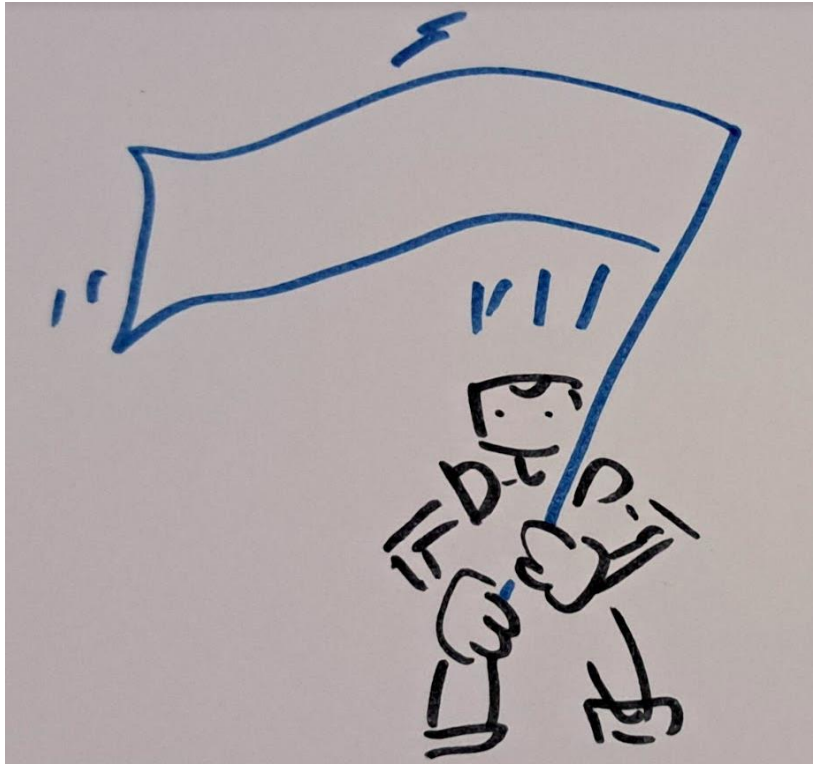


Can also use ArucoMaze functions

Physical tasks / *NAO Challenge*

Joint control – most of the challenge is designing the stick

Physical tasks | *NAO Challenge*

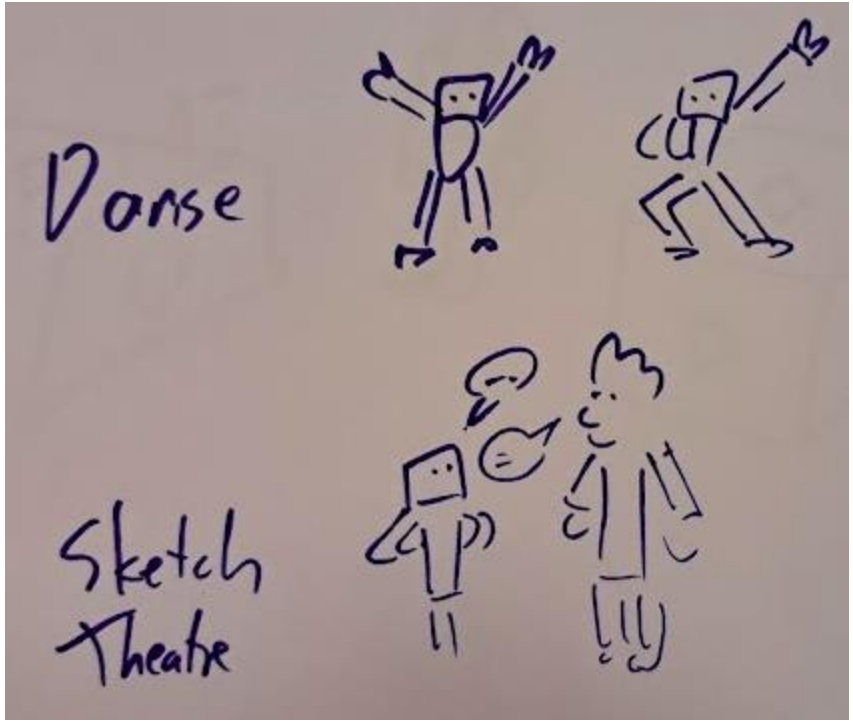


Joint control or animation

Physical tasks / *NAO Challenge*

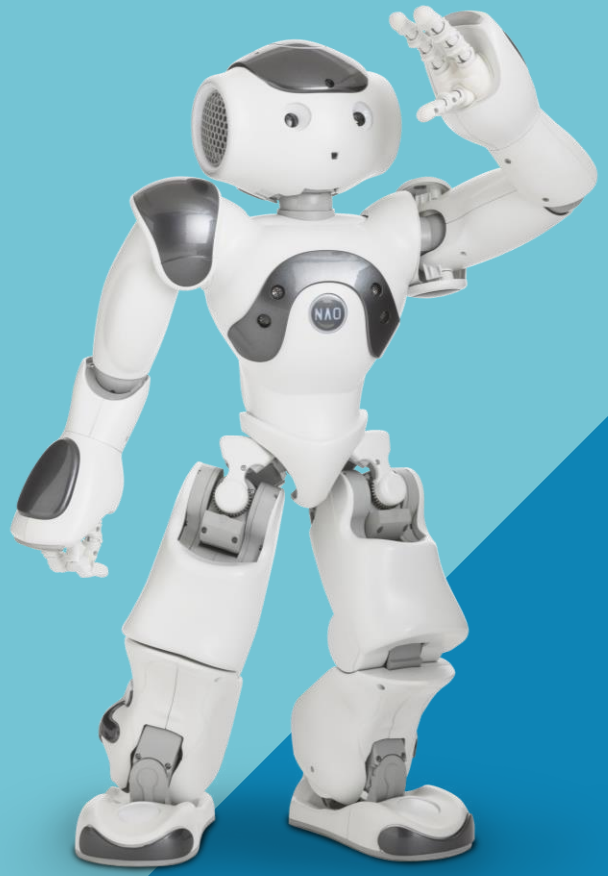
Joint control

Physical tasks / *NAO Challenge*

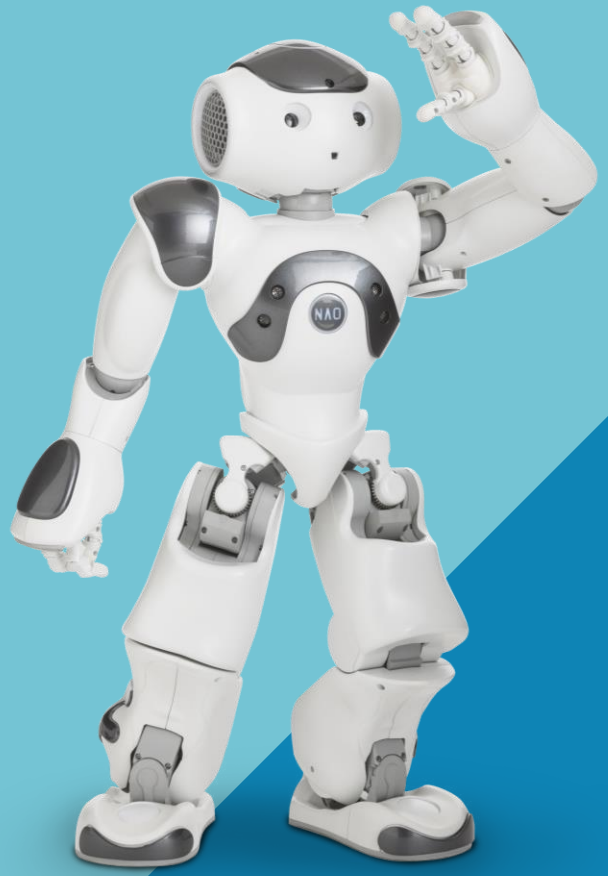


Animation, text to speech, etc.

Thank you!



Appendix



Debug & Test App | Choregraphe

→ Debug by playing the app

→ Your app is uploaded under the name:

".lastUploadedChoregrapheBehavior"

→ After debugging, install and test the app as if it came
from the store:

Finally, package and upload to the store!

Note: you will need to fill-in the properties:

Application:

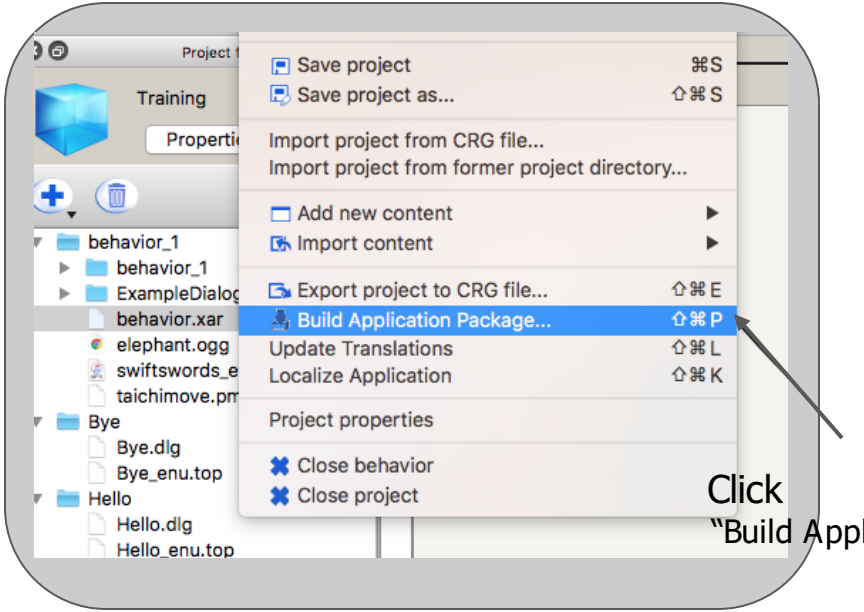
- Name
- ID
- Description…

Behaviors:

- Name
- Trigger sentences

Dialogs:

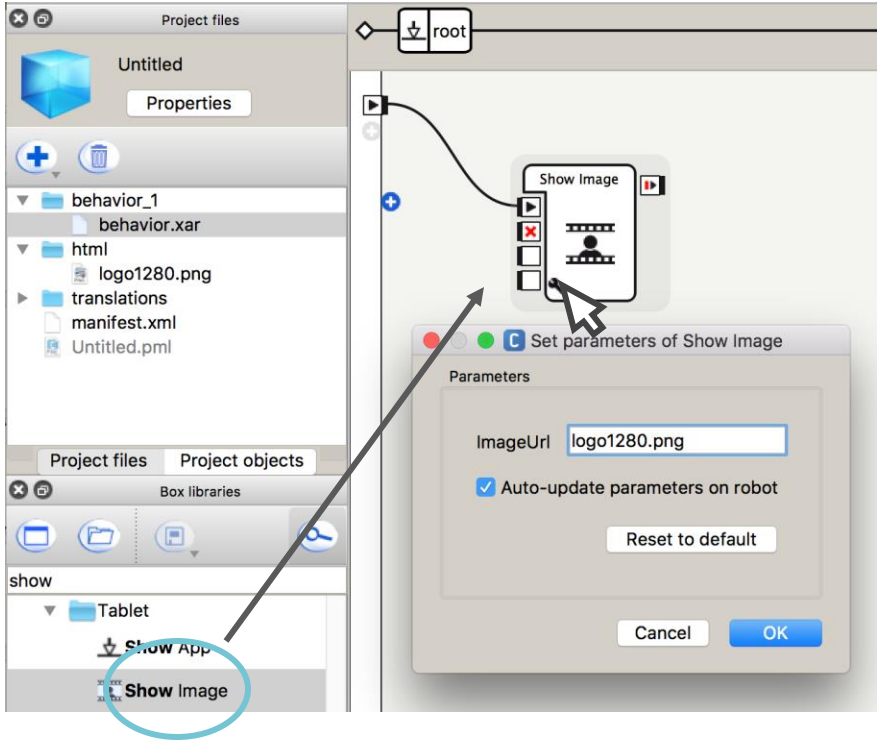
- Availability from autonomous life



Click "Build Application Package…"

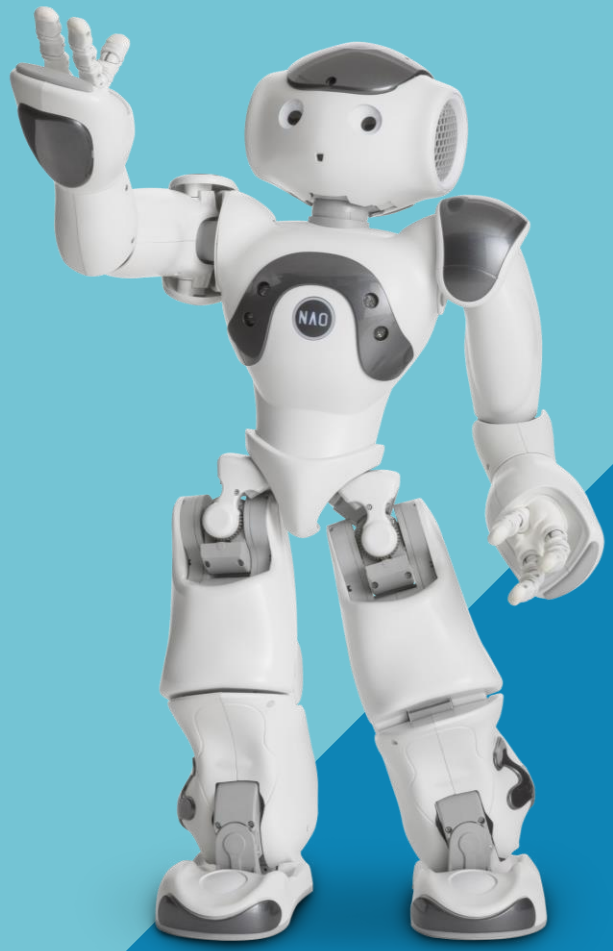
Display your App webpage | Choregraphe

1. Create a new directory: “html”
2. Import an image into this folder
3. Find the box “Show image”
4. Add a link to it
5. Edit the parameter to the name of your image
6. Play the behavior!



Choregraphe - Timeline

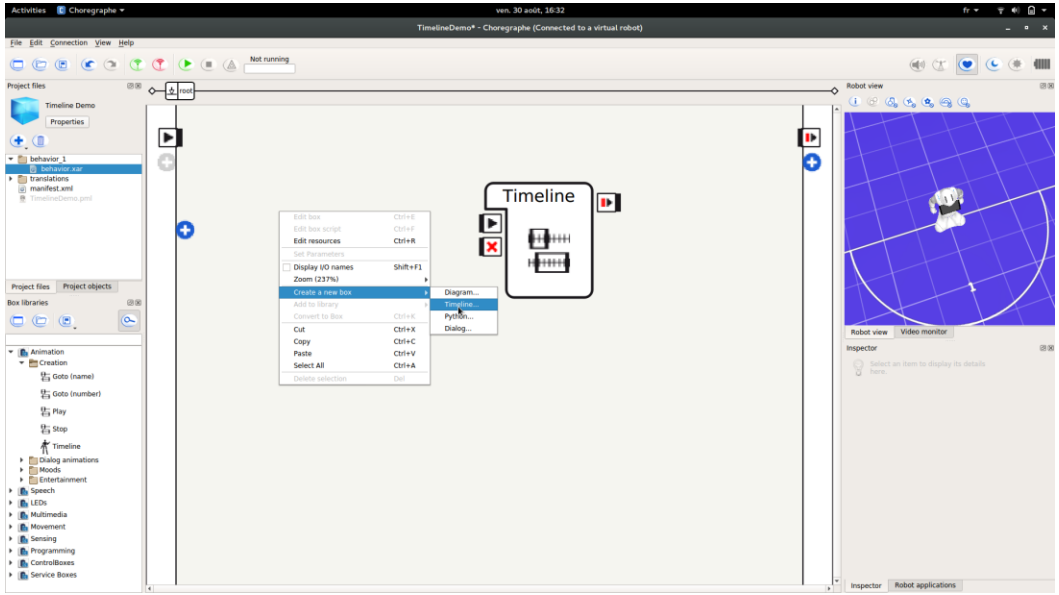
Build custom animations



Build animations with timeline

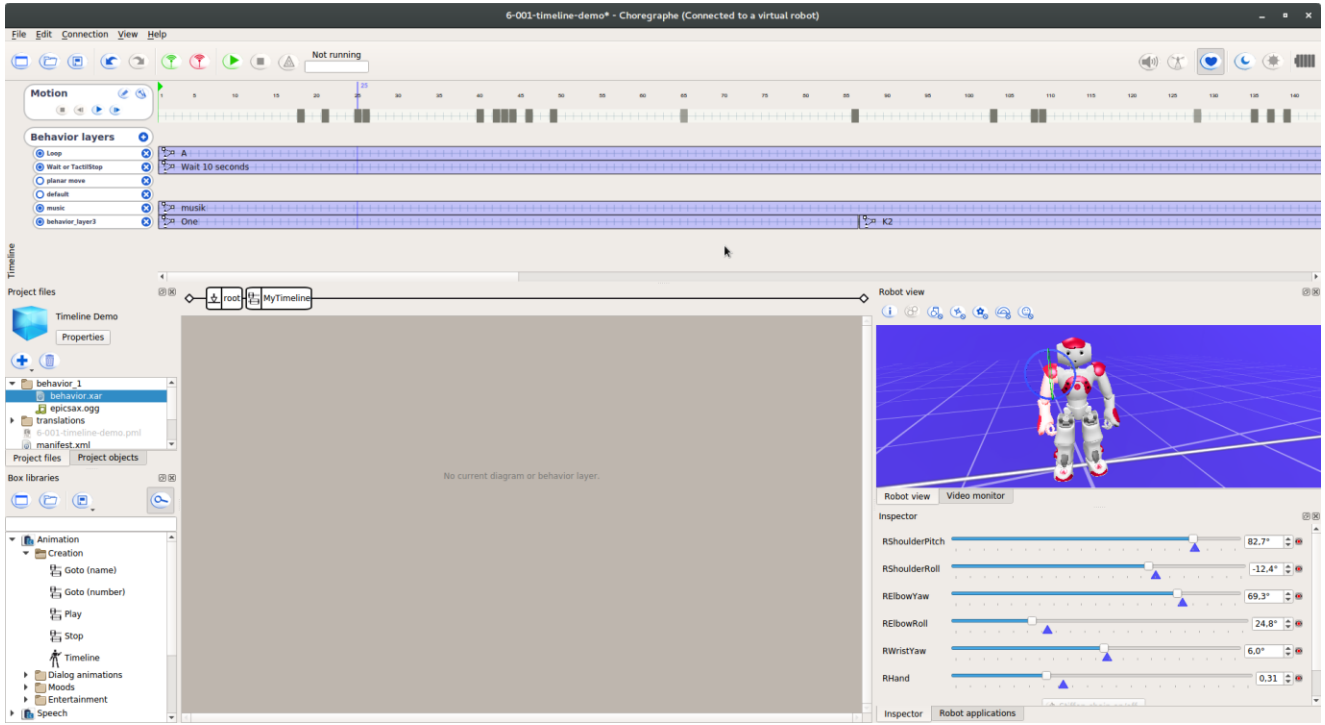
Choregraphe includes a tool to build your own animations called Timeline. To create one:

- Drag and drop a Timeline box from the box library
- Or Right click on the diagram zone and choose Create a new box > Timeline



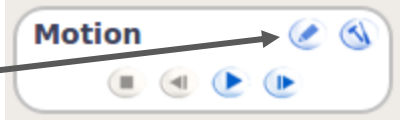
Build animations with timeline

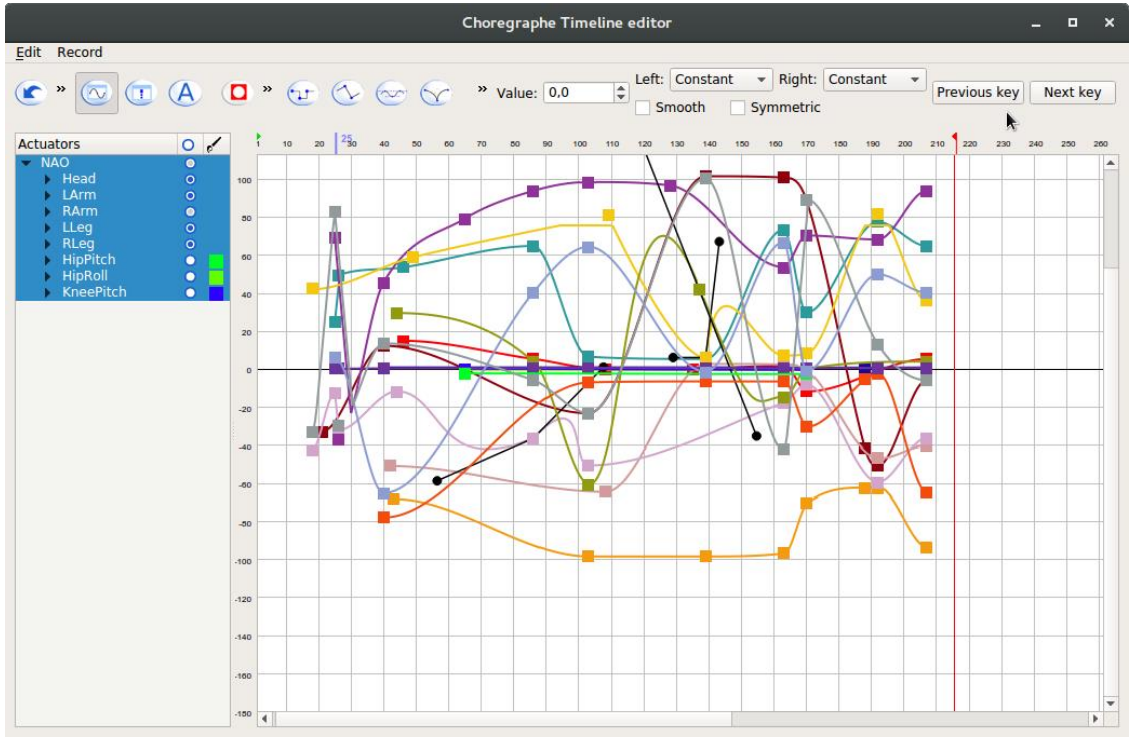
Double click on the timeline box to access timeline edition mode:



Build animations with timeline

From the Motion control you can access Timeline Editor :



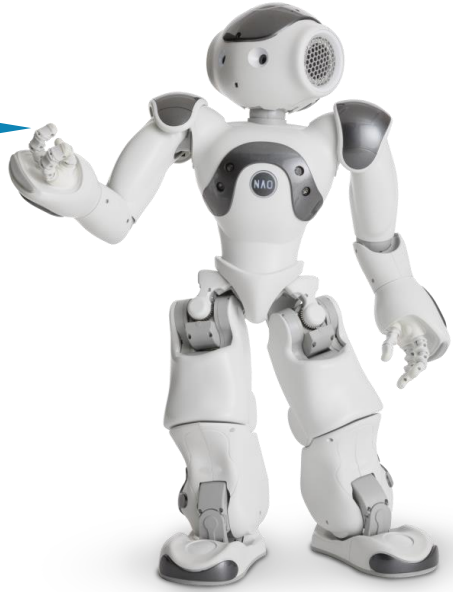


Challenge: Nao is Usain Bolt (the dab part)!

Use timeline to create a dab with Nao!

Do you dab?

Yes I can dab!



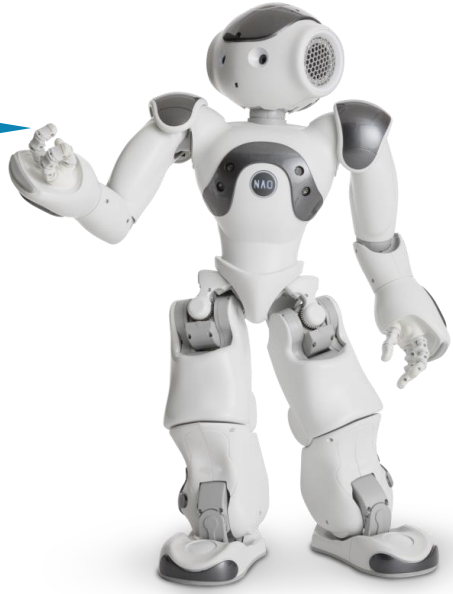
Nao is cool!

Challenge: Nao knows Macarena!

Play music, movements and make you robot turn all together!

Let’s dance!

Okay, I know Macarena!



Ingredients :

- Timeline
- Play box
- Move toward box
- Sound file

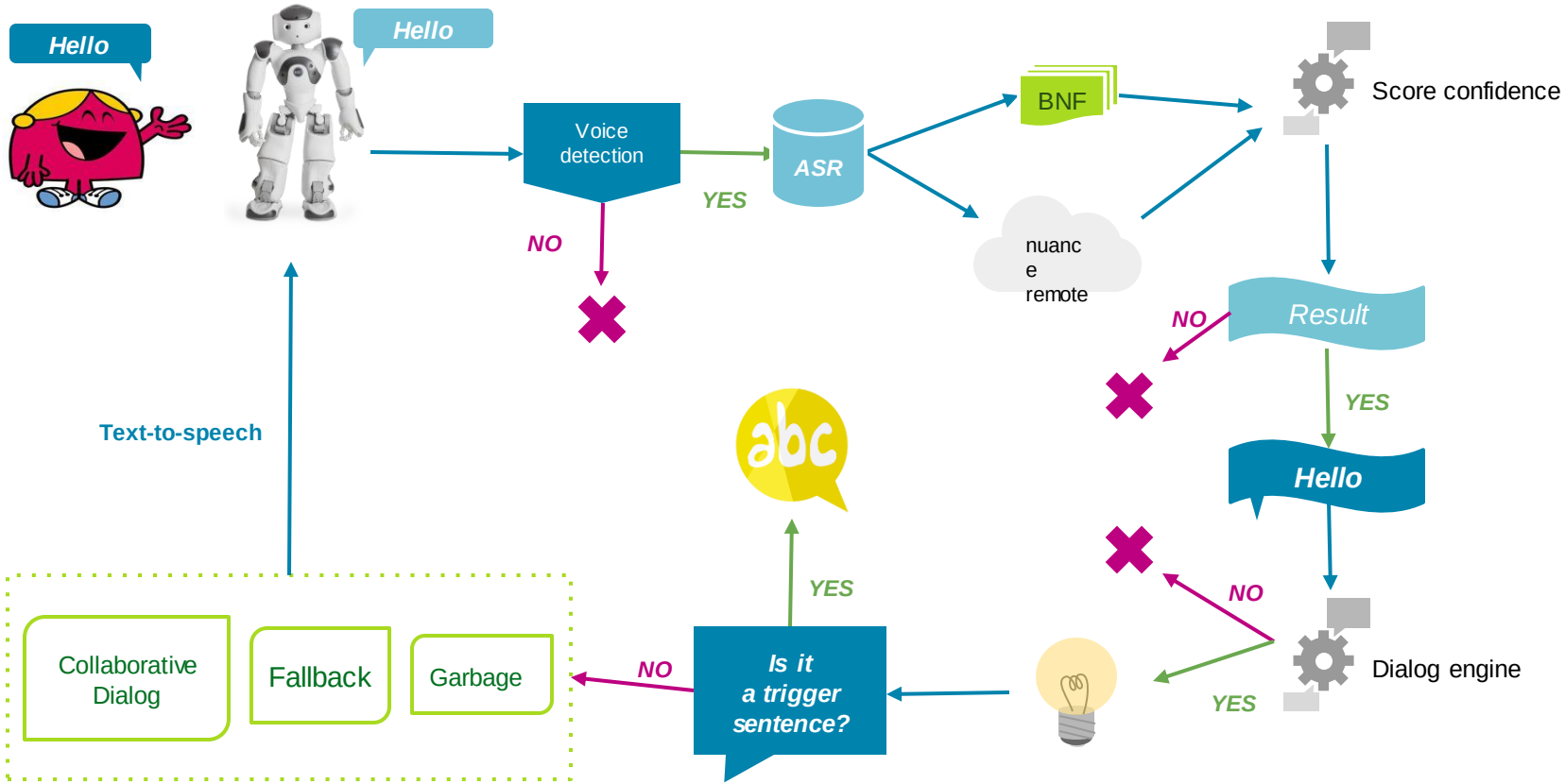
QiChat

Overview

How it works

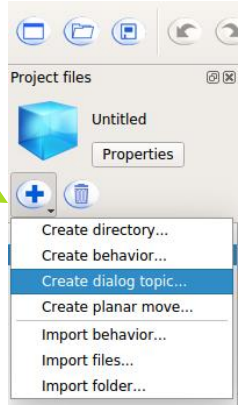
Human in zone 1 (1,5m)

How it works



Let's start:

1. Create a .top file in the Choregraphe project with (+) Create dialog topic...
2. Call this 'hello' and give it the English language
3. Choregraphe will generate a basic conversation for us to play with
   a. A folder "hello" is created which contains
   b. A "hello.dlg" file
   c. A "hello_enu.top" topic file



The name of the topic

A concept is a list of words or phrases which apply to an idea. This is important to make speech feel more natural.

The human input

The robot response

```
topic: ~hello()
language: enu
# Defining extra concepts out of words or group of words
concept:(hello) [hello hi hey "good morning" greetings]

# Replying to speech
u:(~hello) ~hello
```

Dialog Structure | *Basic Informations*

Topic rules u:(input) output

u:(hello) hello human

u:(bye) goodbye, thank you

The robot continuously listens to all triggers

All **rules** are evaluated in parallel

[ ] lists (use space or double quotes “” as delimiters)

u:(Can you [talk speak sing]) [yes absolutely “I do”]

{} optional

u:(Are you a {funky} robot)

Really! Do I look like I am not?

Dialog Structure | *QiChat Syntax*

- https://developer.softbankrobotics.com/nao6/naoqi-developer-guide/naoqi-apis/naoqi-interaction-engines/aldialog/qichat-table-content-3

  https://developer.softbankrobotics.com/nao6/naoqi-developer-guide/naoqi-apis/naoqi-interaction-engines/aldialog/qichat-table-content-0

Dialog Structure | *Priorities in Dialog*

If a u: rule or proposal has **subrules** (u1, u2...), they **have the priority**.
The subrules are called the **scope of the rule**.

u: (hello robot) hello human, how are you ?
u1: (fine) good
u1: (bad) I hope I can help you
u1: (talk about something else)
ok, do you want me to tell you a joke
u2:(yes) ...

```
u:(input1) answer                    Scope of the rule u:
   u1:(input2) answer
   u1:(input3) answer      Scope of the rule u1:
      u2:(input4) answer
         u3:(input5) answer
      u2:(input6) answer

proposal: sentence
   u1:(input7) answer
   u1:(input8) answer
```

proposal: %tag
Do you want to dance or to play ?
u1:(dance) ok

Note : indentation doesn't matter but is a good practice

Main functions in QiChat

%TAG
^nextProposal
^gotoReactivate(tag)
^gotoRandom(tag)

^stayInScope

^first [
“$var==1 sentence 1”
“sentence 2”
]
^rand [“sentence 1” “sentence 2”]
[“sentence 1” “sentence 2” “sentence 3”]
^clear(var)

Variables in QiChat / *QiChat Syntax*

Local variables

Catch some part of human input and use it in robot answer: _ and $ characters

u:(i need a _**[bike lime uber]** to go to the **_[“swimming pool” airport]**)

I am going to help you finding a **$1** to go to the **$2**

Global variables

Create a variable using =

Use the variable using its name and $ sign

Erase with ^clear()

u:(do you have _**[shoes trousers]**) **$item=$1** You are looking for **$1** right ?

u1:(yes) What kind of **$item** do you want ?

u1:(I changed my mind) no problem **^clear(item)**

Remote ASR

You can use _* syntax in an input

u:(my name is _*****)
Hello **$1**

This will use the remote ASR to process Speech recognition
Constraints:
network needs to be active
amount of requests available 5.000 requests/day/robot
Only available in some languages
Option needs to be activated (sales, customer care)

QiChat refresher

```
u:(I {really} like _[tea coffee]) Good to know
$drink=$1

u:(My name is _*) Pleased to meet you $1 $name=$1

u:(Do you know me?)
 ^first[
   "Yes, you are $name and you like $drink"
   "Yes, you are $name"
   "I know you like $drink"
   "No, I don't know you"
 ]
```

Now that we have a topic, let's make it a bit more interesting

Time to test this on me !

QiChat refresher

```
u:(I {really} like _[tea coffee]) Good to know
$drink=$1

u:(My name is _*) Pleased to meet you $1 $name=$1

u:(Do you know me?)
 ^first[
   "Yes, you are $name and you like $drink"
   "Yes, you are $name"
   "I know you like $drink"
   "No, I don't know you"
 ]
```

Using **underscore _** creates a special **variable**, **$1**, that you can assign to another variable

Use it with the **wildcard *** to get any inputs.

Use **^first** to get the first line with no empty variables.

Concept & Lexicon

- dialog concept qichat web link

  Concept definition

  concept:(yes) [yes yeah yep ok cool]
  concept:(no) [no nope “not at all”]

  Concept call

  include: lexicon_enu.top
  proposal: Do you like sport?
  u1: (~yes) Okay, Let’s talk about sport! ^nextProposal()
  u1: (~no) Okay! ^topicRandom()

Add animation

- Functions
  ^start()
  ^wait()
  ^stop()

  Animations are located in the “Animation Library” package in Robot Applications tab

  Exercise: ask your robot to dance ;-)

Play with TTS

- altexttospeech-tutorial web link

Punctuation , . ! ? : ...
Pause \pau=value\
Pitch \vct=value\ (50 & 200) 100 by default
Speaking rate \rspd=value\ (50 & 400) 100 by default
Reset \rst\
\style=joyfull\ \style=neutral\ \style=didactic\

play with TTS ;)

Challenge: Mad libs!

Make a little word game! Use concepts, lexicon concepts, qiChat functions, ...

Let's play!

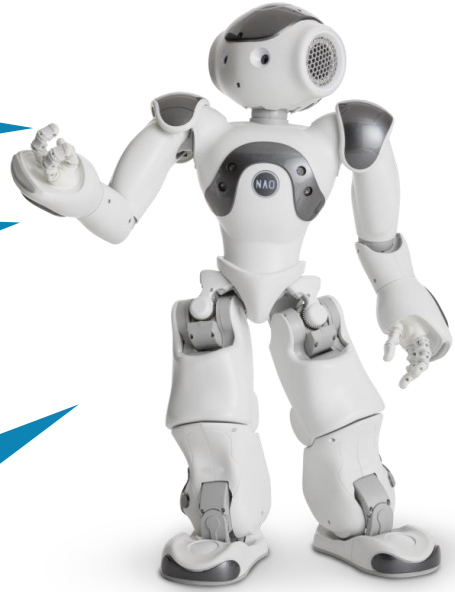
Okay, give me a noun.

Mushroom

And an animal

Rabbit

New start-up: It's like Uber, but selling mushrooms to rabbits.



Challenge: Guess a number

Make a little game, where you make NAO guess a number:

Think of a number and tell me when you're ready.

I'm ready !

Is it ten?

No it's smaller

How about five?

Still smaller

Three then?

No, larger

Four?

That's right!

Challenge: Where should I go?

Make a little game, where you make NAO store movement instructions and run when ready:

Where should I go? Say “forward”, “backward”, “left” or “right” and I will play the movements after you say “Ready”.

Forward

Ok foreward (sound)

Left

Ok left (sound)

Ready

Let’s go

Nao will make the movement walking through the path you told him.

The (real) end