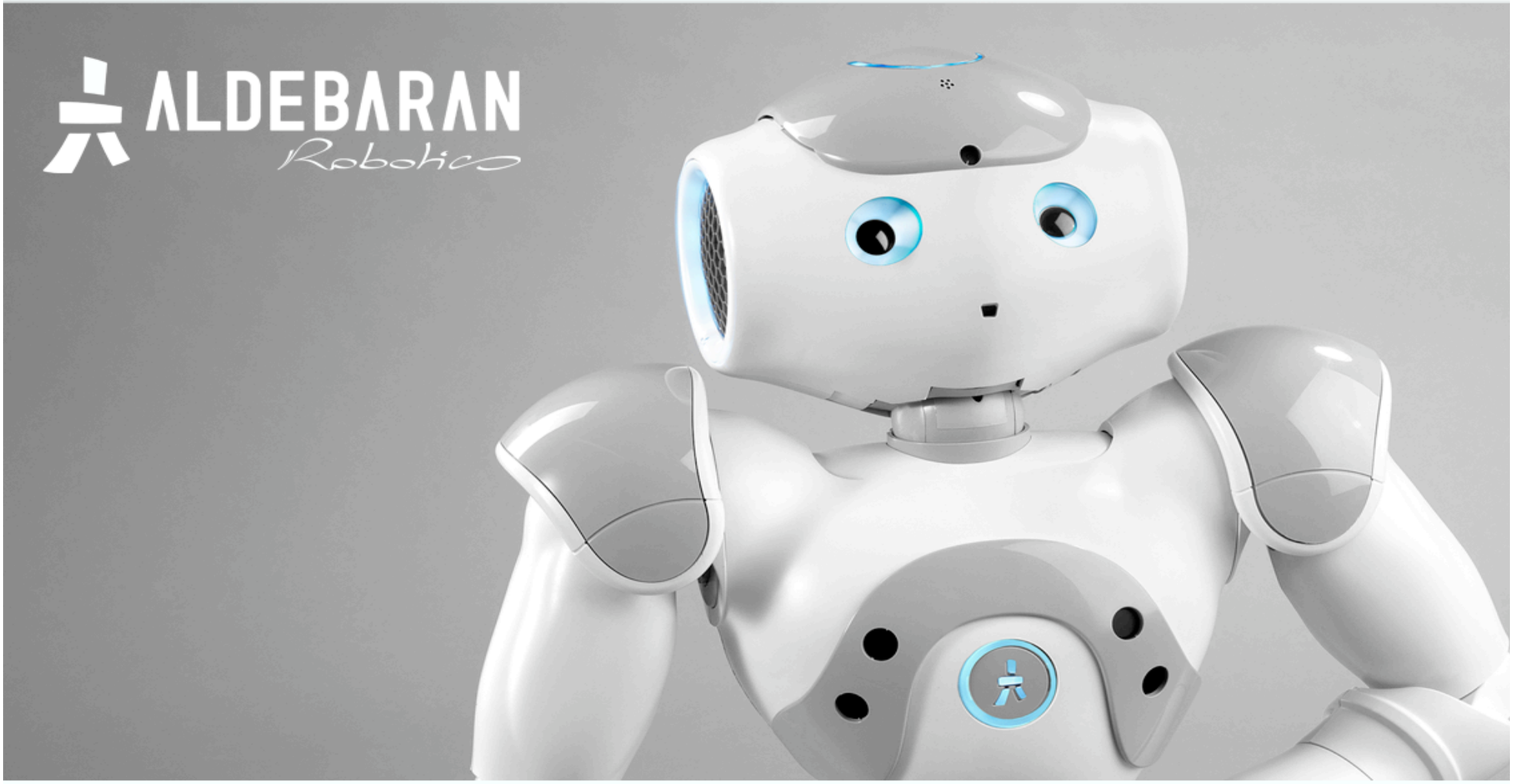
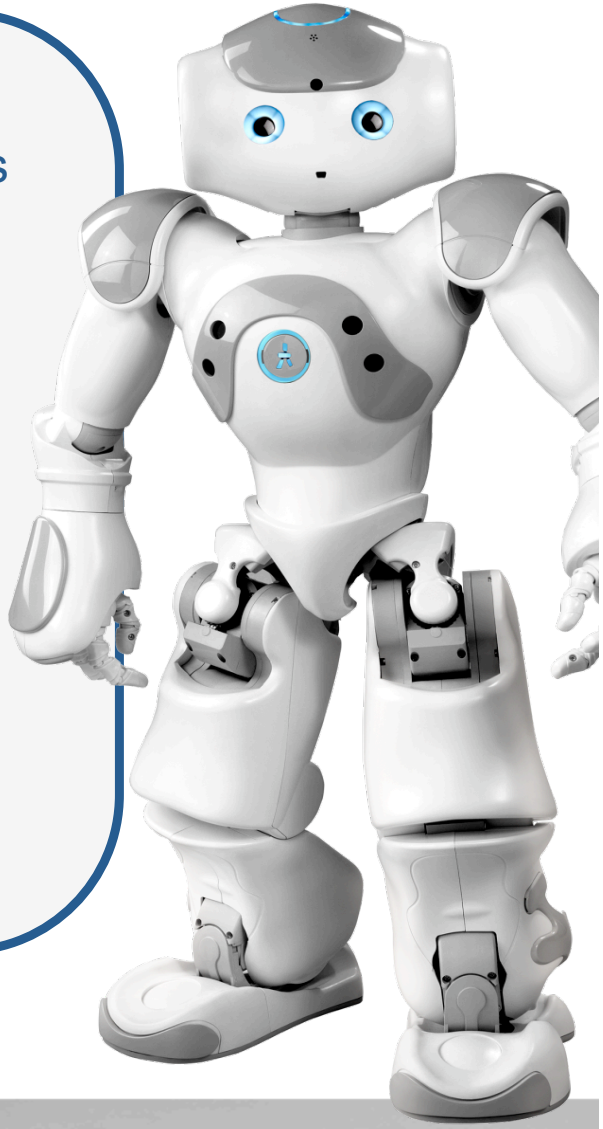


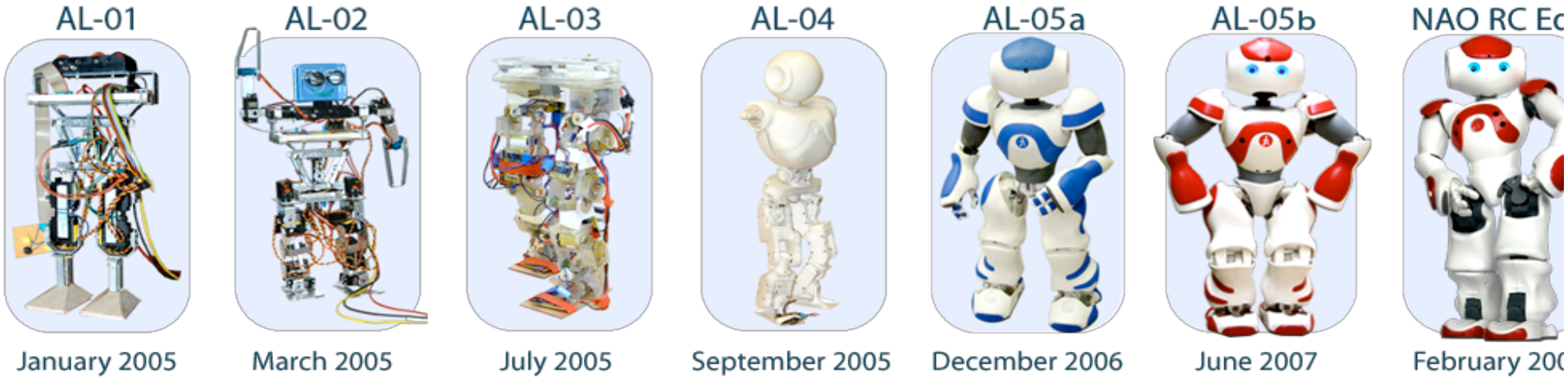
Aldebaran Robotics’ NAO

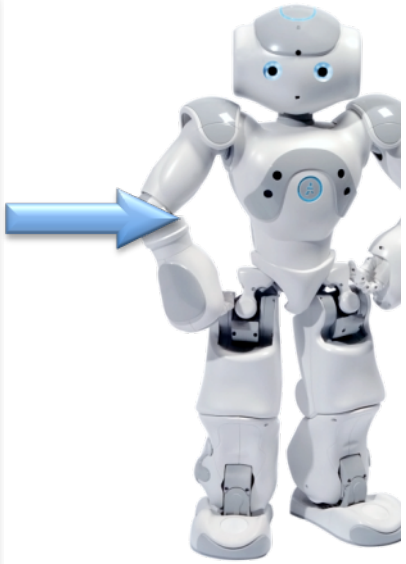
Who we are, in a nutshell

- Founded in 2005, **European** company based in Paris
- Goal : spread **humanoid robots** for :
  - Personal Assistants, home companion
  - Research and Education
- **900** NAOs sold in **30** countries
- **World leader** in BtoB humanoid robotics
- Working closely with **R&D labs** and **Educational** Institutions



NAO project: design stages





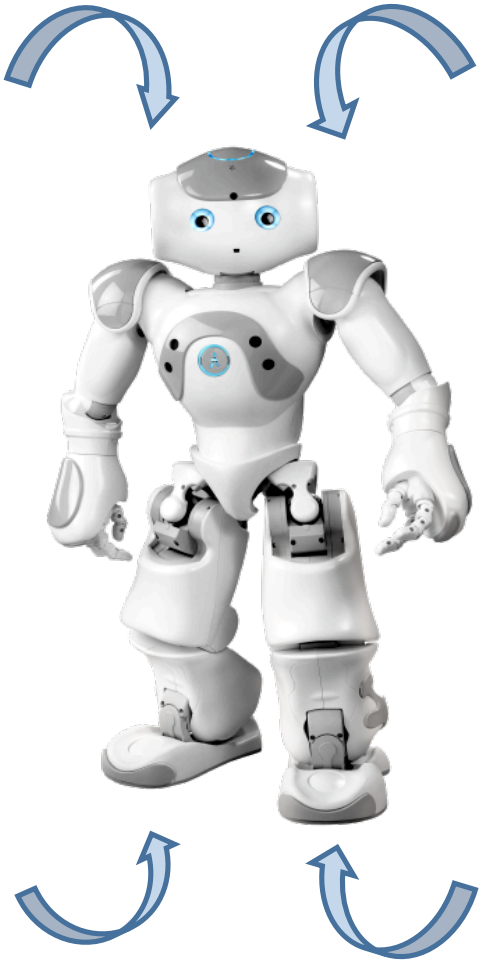
What can NAO do?

Move

- **25 Degrees of Freedom**
- Smooth and precise **coreless motors** (Maxxon)
- Controlled with software

Sense

- **2 cameras**
- **4 microphones**
- **8 FSRs, 2 Bumpers**
- **DCM**
- **2 Sonars**



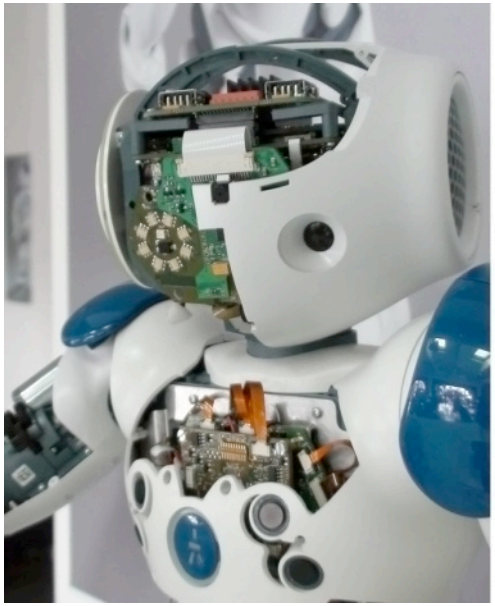
Communicate

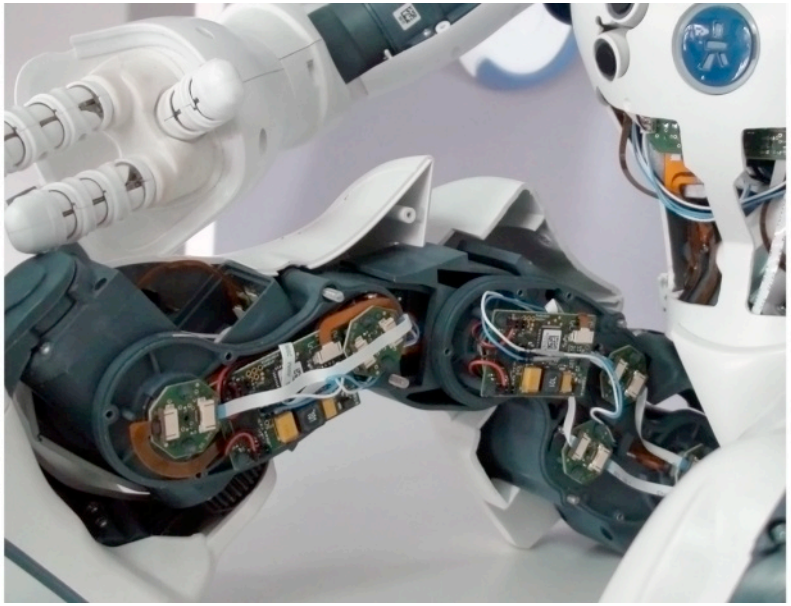
- **2 loudspeakers**
- Multiple **LEDs**
- **Tactile** sensors, **prehensile** hands
- **Infrared** sensors
- **WIFI** connection

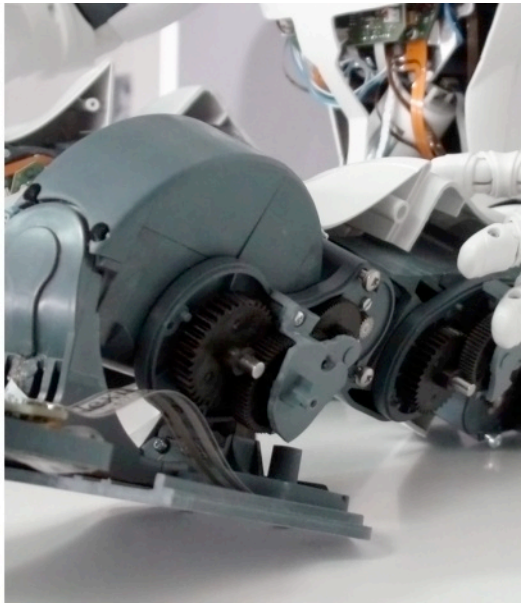
« Think »

- **Geode 500 Mhz CPU**
- 256 MB SDRAM
- 2 GB Flash Memory
- **Software suite** + **SDK** to program Nao

Inside NAO







- Head with onboard computer, Leds and 2 cameras

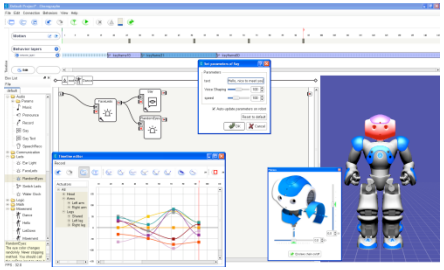
- Chest electronic board with sensors and the ARM9

- Magnetic Rotary Encoders and motor controller

- Gears and Force Sensing Resistors

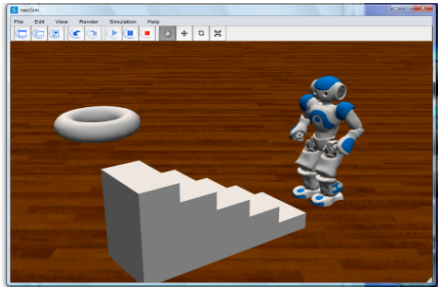
Our Software Suite

More than a software suite, a **comprehensive programming environment**



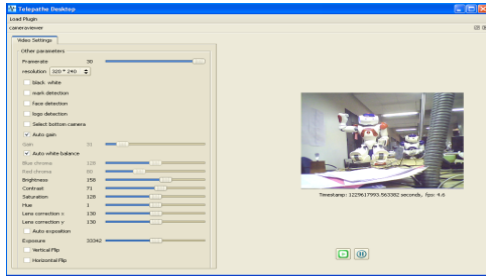
Choregraphe

- Ergonomic and **user-friendly** interface
- Drag and drop **behavior** boxes in the **flow diagram**



NAOsim

- Official **simulator** for NAO
- Quickly **test** new robotic behaviors & applications



Monitor

- **Feedback** of what NAO is seeing and feeling
- **Ergonomic** interface to access the data from the ro **sensors**

SDK

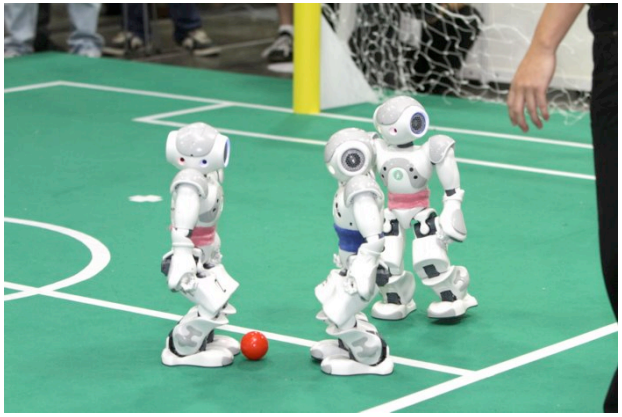
- **Embed** modules you hav created into your robot in or to create **elaborate** behavic for NAO
- **Compilation** and **debugg** tools.

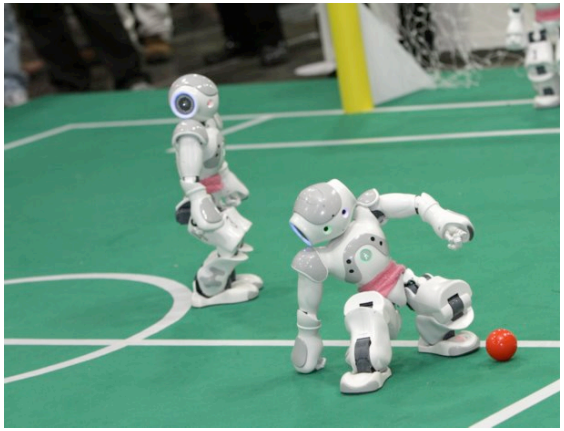
Programming NAO

Many possible ways to access NAOqi APIs :

Languages	Running on…	OS	Remarks	Tools
Choregraphe			Python code running locally on the robot	Choregraphe
Python URBI			Communications with the robot may be **slow**.	Scite…
C++				Visual Studio 2005/2008, Xcode, GCC...
			Cross compilation available on Linux (or Linux virtual machine) **Real-time is possible**	Eclipse
.NET			Tools: Visual Studio	

Standard Platform for Robocup

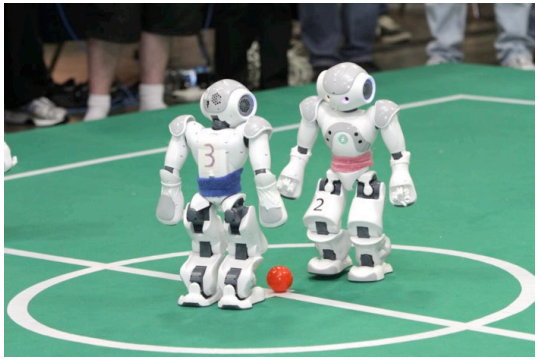




350 teams, multiple leagues, +3000 students

SP League : each team uses exact same hardware

SONY's AIBO was the standard platform until 2006



24 teams from 18 countries
used NAO during **RoboCup** 2010 in **Singapore**

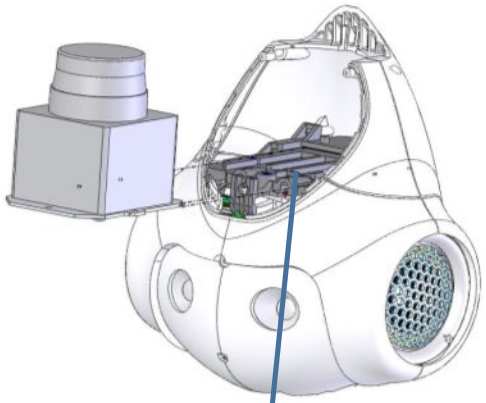
Laser Head

Removable door

- Special head with Hokuyo Laser Scanner

URG-04LX Laser

Detection range	0.02 to approximately 4m
Scan angle	240°
Scan time	100msec/scan (10.0Hz)
Resolution	1mm
Interface	USB 2.0, RS232



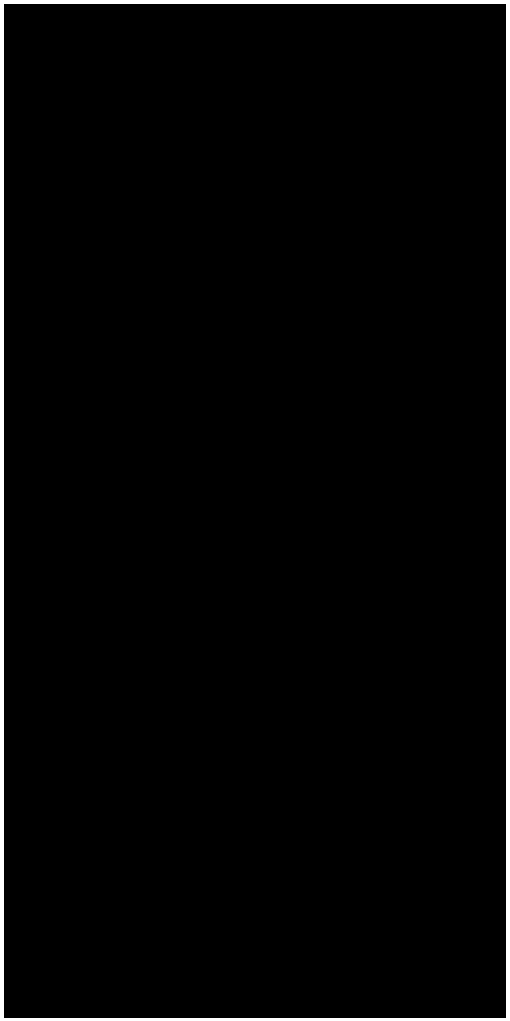
Removable Laser

Perfect for mapping, planning, localization

Romeo Project

- Ambitious research project
- Objective :

  Develop a **humanoid robot** which can serve as a **Personal assistant**
- Prototype due to Spring 2011

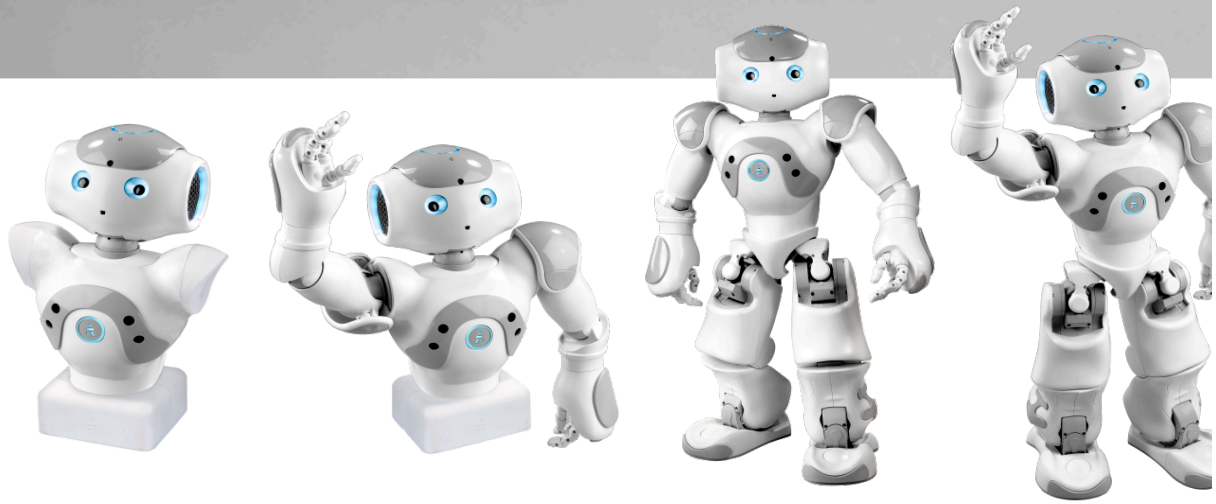


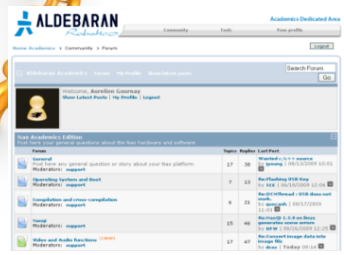
Partners :

Romeo Project

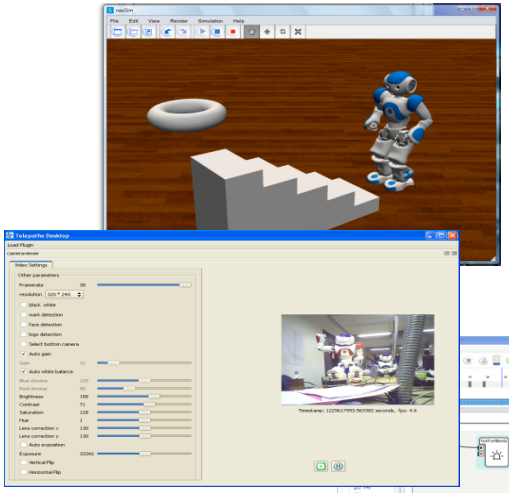
Our Offer

A full range of products

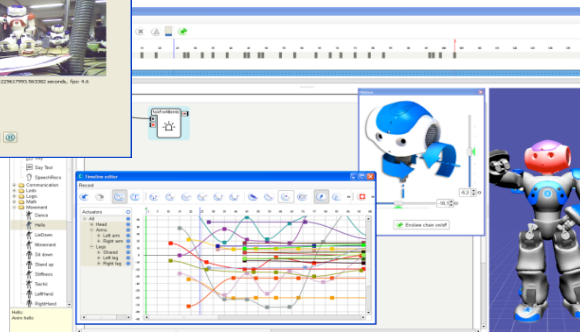




Dynamic community of users

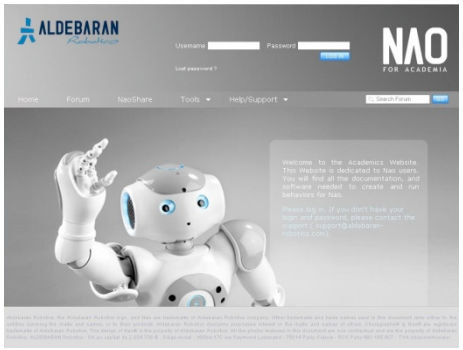


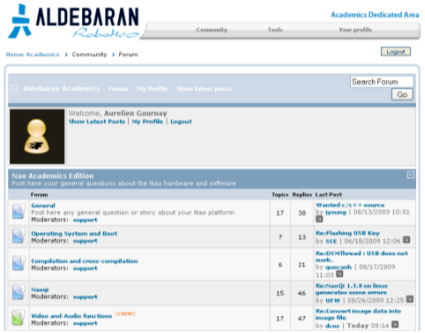
A whole Programmin Environmen



A dynamic community of users

NAO Academia, dedicated to NAO users



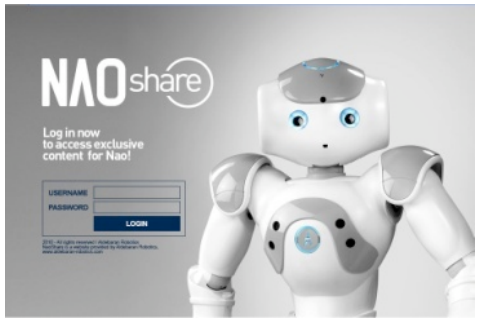


A dedicated forum:

- **Community**: be in touch with other NAO owners
- **Support**: talk with Aldebaran Robotics Support and R&D teams

NAOshare

Web-based sharing application of content related to NAO



Online Documentation

Thank you!

…and see you soon

Autonomous and Mobile Robotics

Prof. Giuseppe Oriolo

Introducing NAO

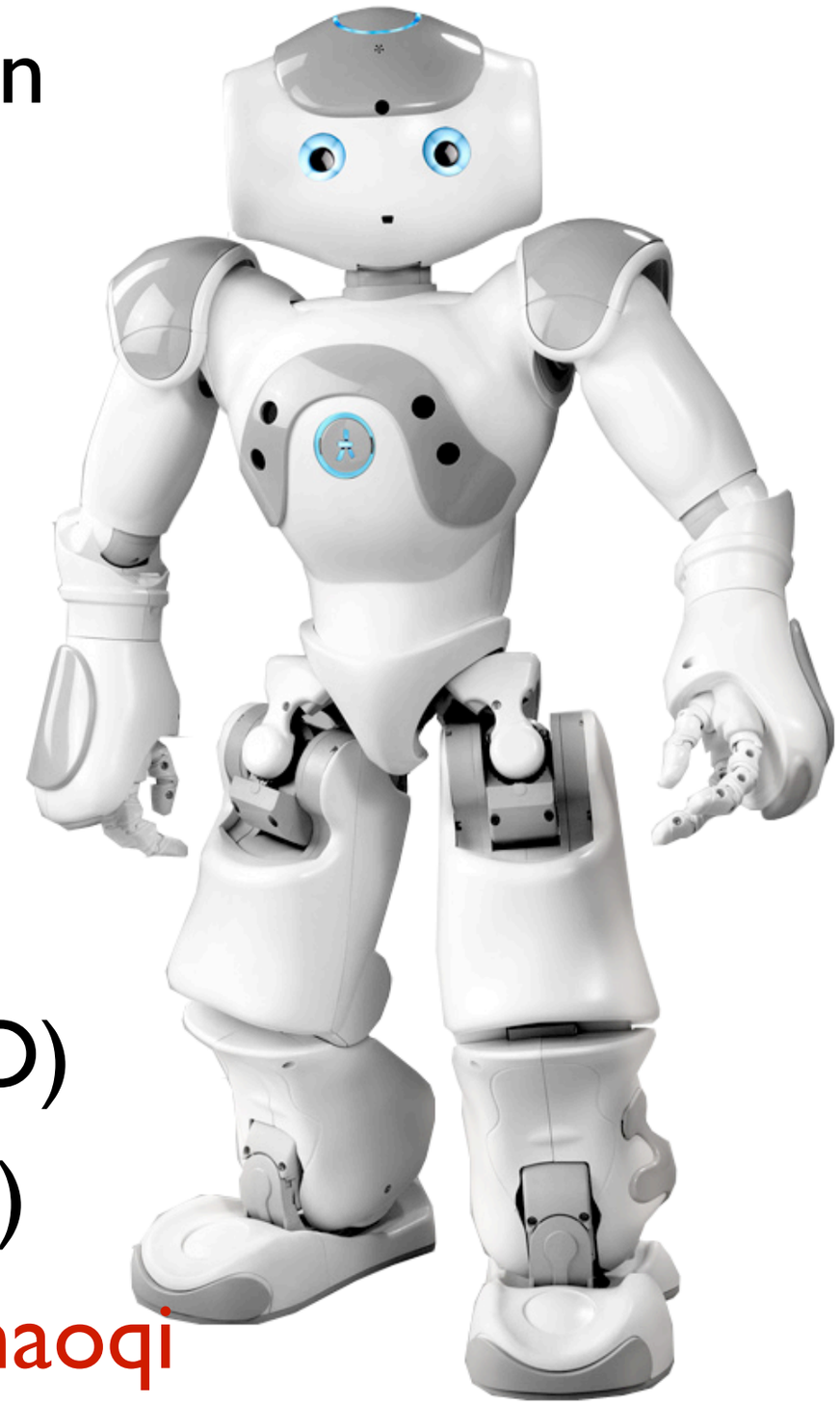
(slides prepared by Antonio Paolillo)

Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti

Sapienza
Università di Roma

hardware

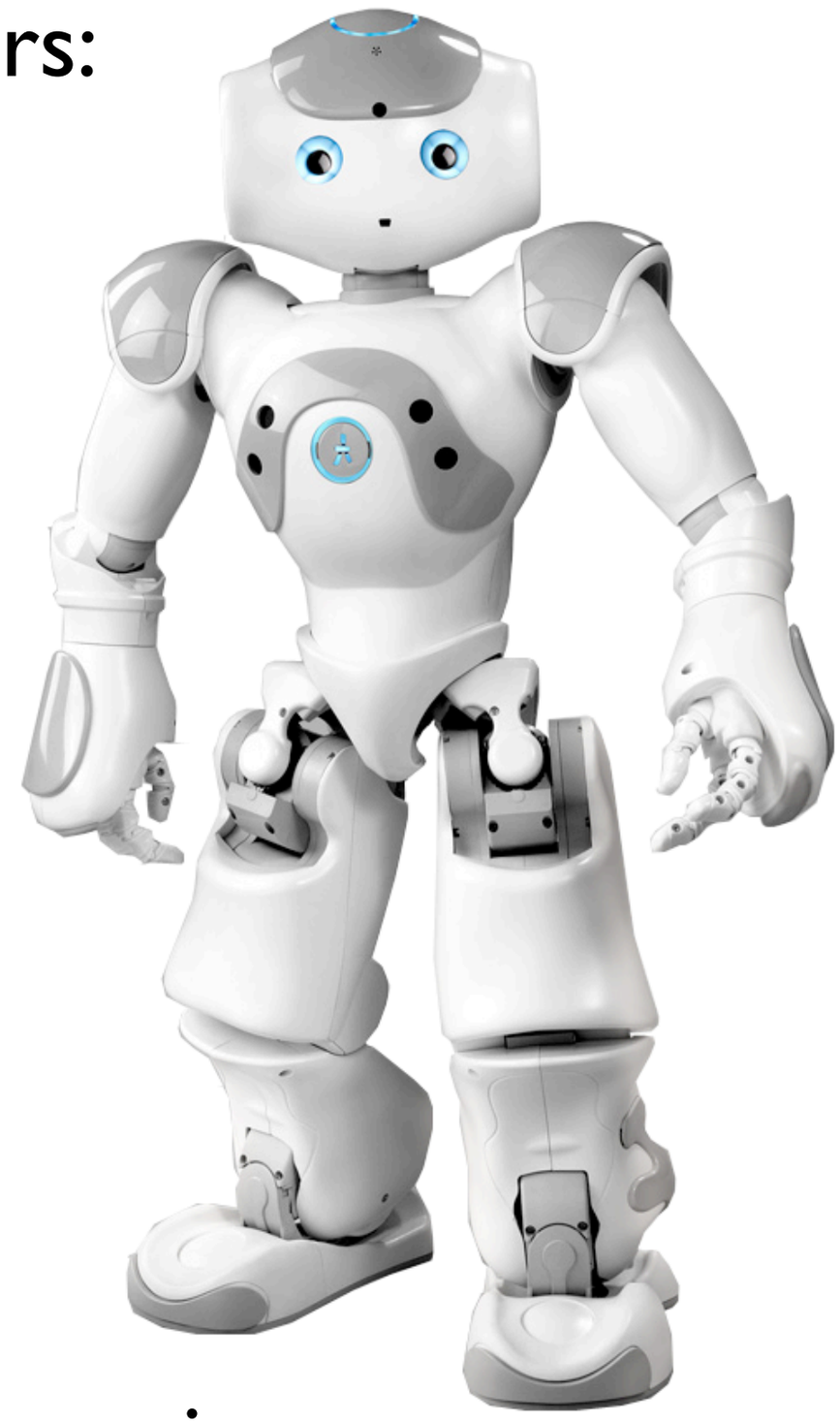
- made by the french company aldebaran
- 25 degrees of freedom
- height = 57 cm
- weight = 4,3 kg
- ATOM Z530 1.6GHz CPU
- 1 GB RAM, 2 GB flash memory
- wi-fi connectivity and ethernet port
- linux 32 bit with NAO OS (OpenNAO)
- fully programmable (C++ for example)
- supported by a software framework: naoqi



hardware

NAO is equipped by a long list of sensors:

- 2 loudspeakers
- 4 microphones
- 2 CMOS digital cameras (30Hz)
- LEDs
- encoders to the joints (100Hz)
- gyrometers and accelerometers
- 2 bumpers
- 2 sonars
- 2 infrareds
- tactile sensors

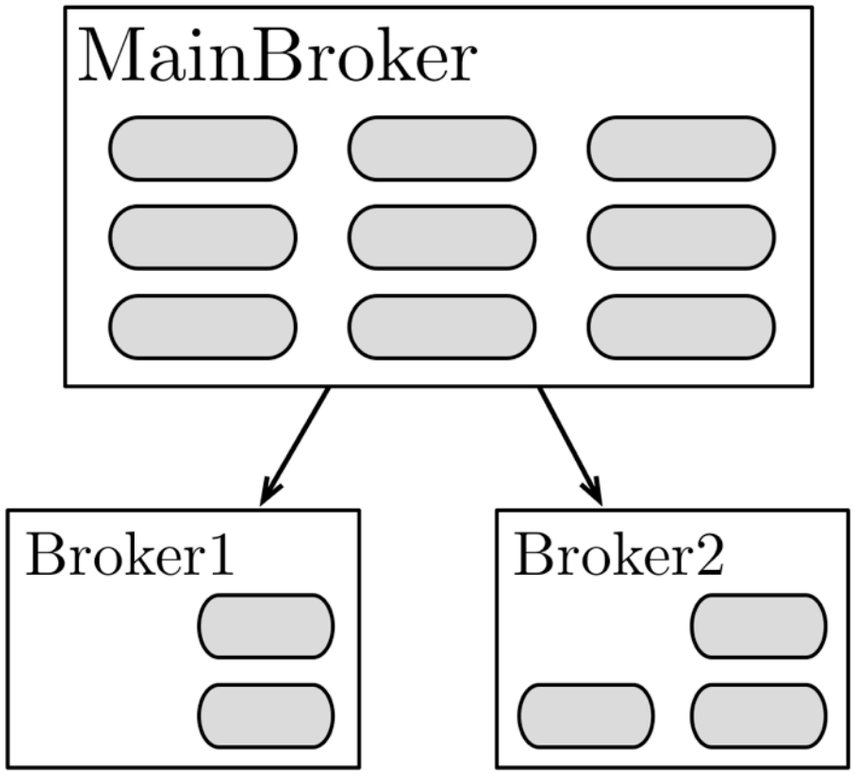


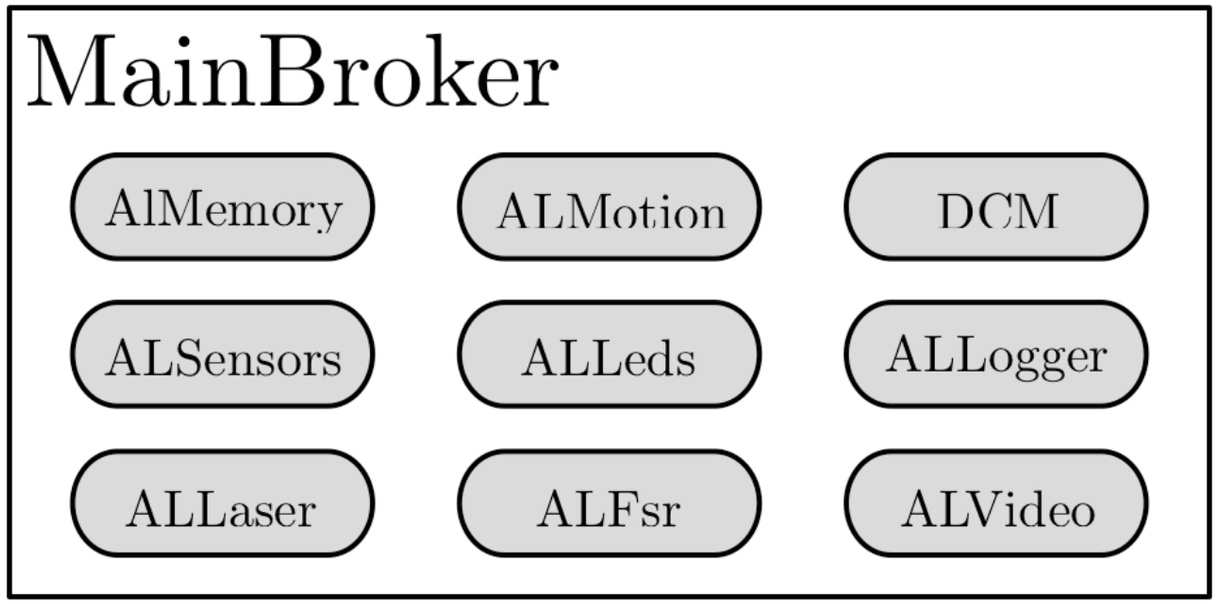
naoqi provides APIs for the motion and sensing

software framework: naoqi

- naoqi is a robotics software framework which allows:
  - parallelism
  - synchronization
  - events
  - resources
- modular structure
- each module has a functionality
- several modules can communicate each other
- software communication is possible thanks to
  - broker
  - proxy

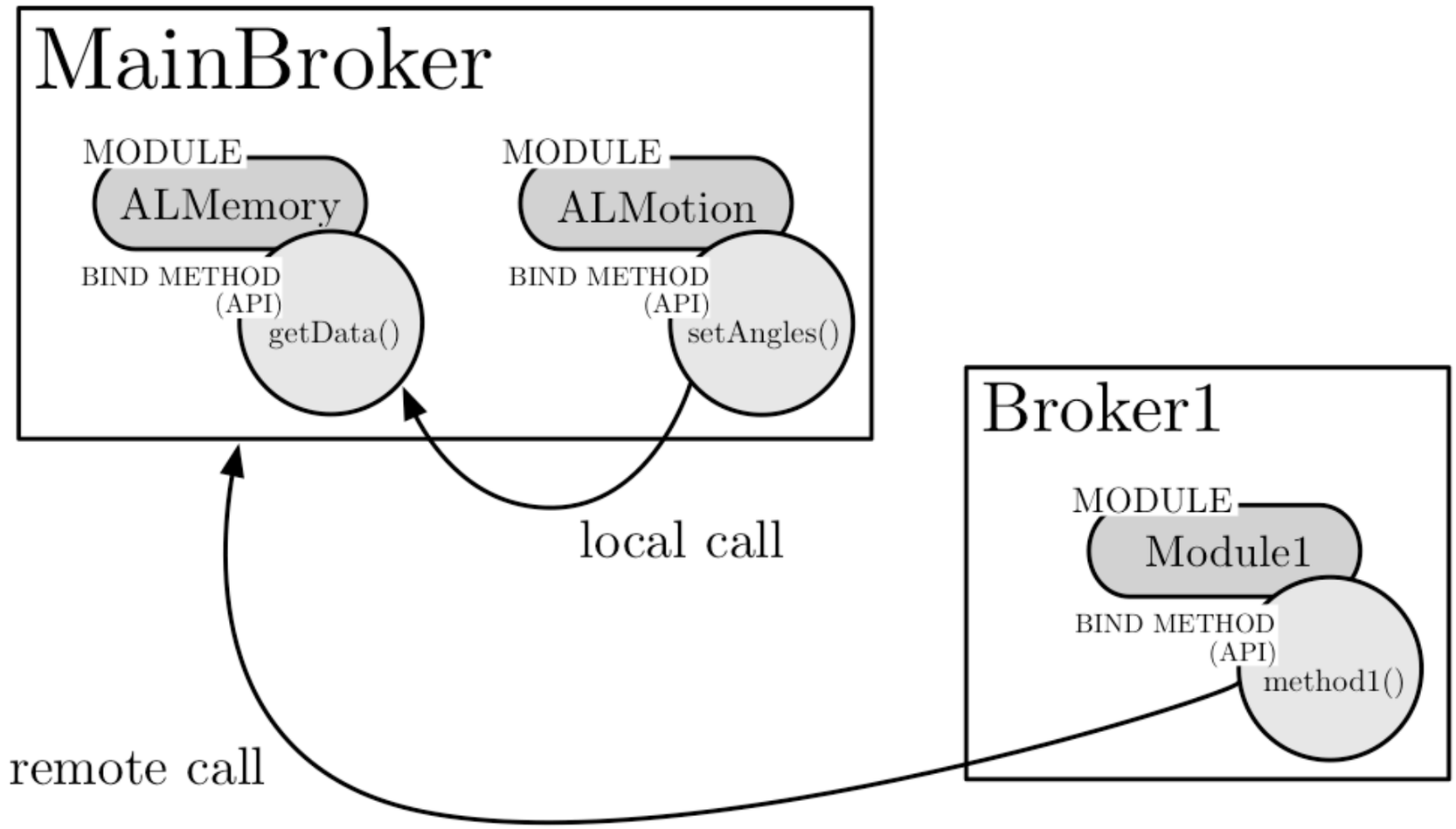
broker





- it's a binary which runs independently and is attached to an IP address
- run on the robot or/and on a computer
- a set of brokers can be structured as a tree
- an application can be made by more brokers (to overcome computational problems)
- functionalities of each broker are given by modules
- each module has special methods (API)
- broker manages messages among modules

proxy



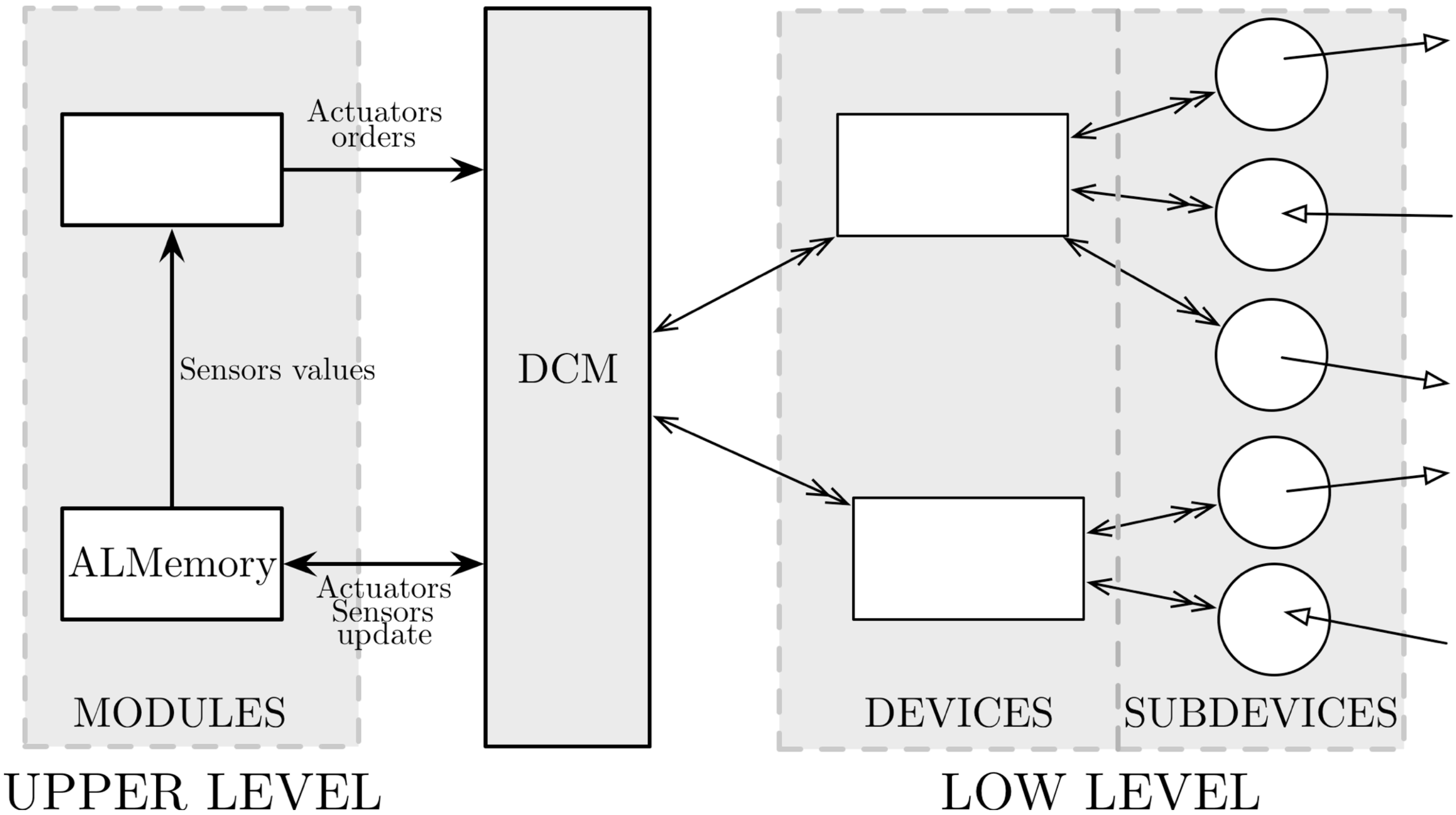
- it allows to use part of code implemented in other modules easily
- it is possible to call methods of a module which
  - belongs to the same broker (local call)
  - belongs to another broker (remote call)

low level programming

- naoqi API use is recommended by Aldebaran (and it is very simple!)
- but, in order to have:
  - direct access to the robot devises
  - fast access to the memory
  - fast execution of the commands

  it is needed to program the robot at low level
- low level programming is more laborious but allows an absolute control of the robot
- DCM (Device Communication Manager) used

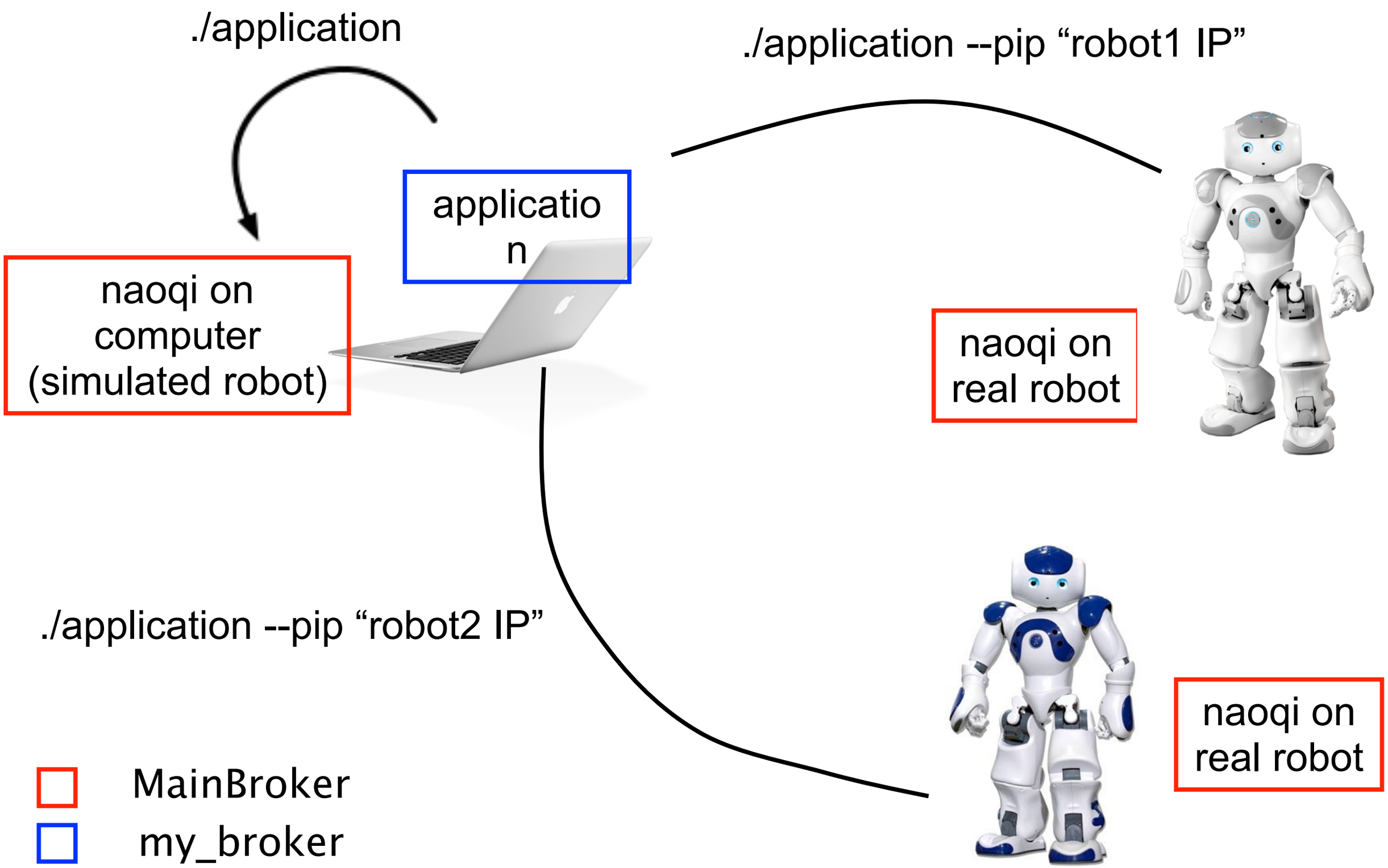
device communication manager



nao programming

- how to create a naoqi module from scratch
  - use the qibuild tool
    - you can generate automatically a ready made module
    - chose name module
  - write some code
  - compile or cross-compile using cmake
    - compilation ⇒ creation of an executable (remote module)
    - cross-compilation ⇒ creation of a library (local module)
  - load the module in naoqi
    - if it is a library, it has to be added in the autoload.ini file
- how to launch the module
  - executable: ./module-name --pip <IP> --port <PORT>
  - library: automatically launched at naoqi start-up.

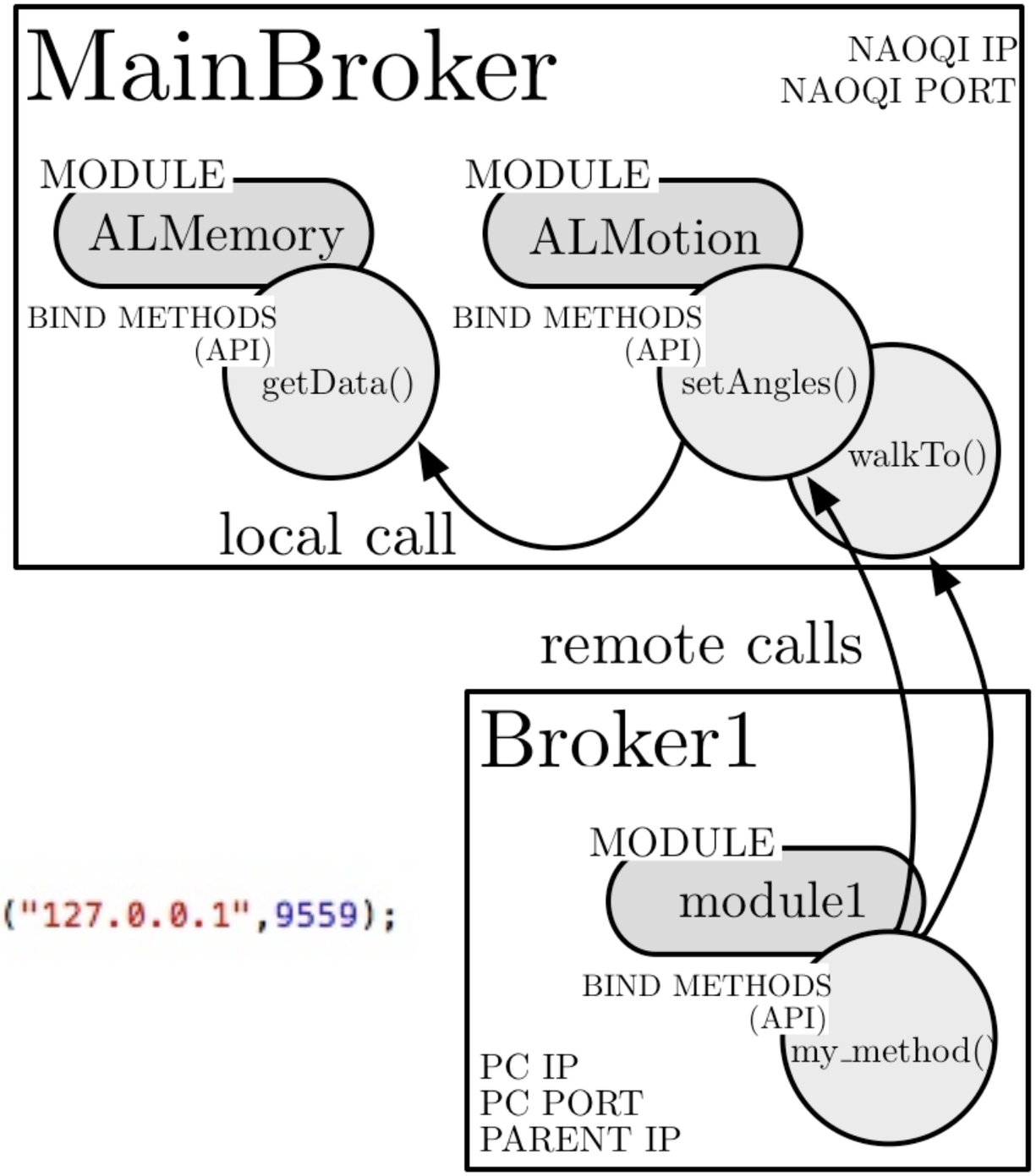
launch of executable



an example

```
ALCALL int _createModule( AL::ALPtr<AL::ALBroker> pBroker )
{
  // init broker with the main broker instance
  // from the parent executable
  AL::ALBrokerManager::setInstance(pBroker->fBrokerManager.lock());
  AL::ALBrokerManager::getInstance()->addBroker(pBroker);

  AL::ALModule::createModule<module1>( pBroker, "module1" );

  return 0;
}
```

```
void module1::my_method(){

  AL::ALMotionProxy *motion = new AL::ALMotionProxy("127.0.0.1",9559);

  motion->walkTo(0.5,0.0,0.0);

  motion->setAngles ("HeadYaw",0.4,0.2);

}
```

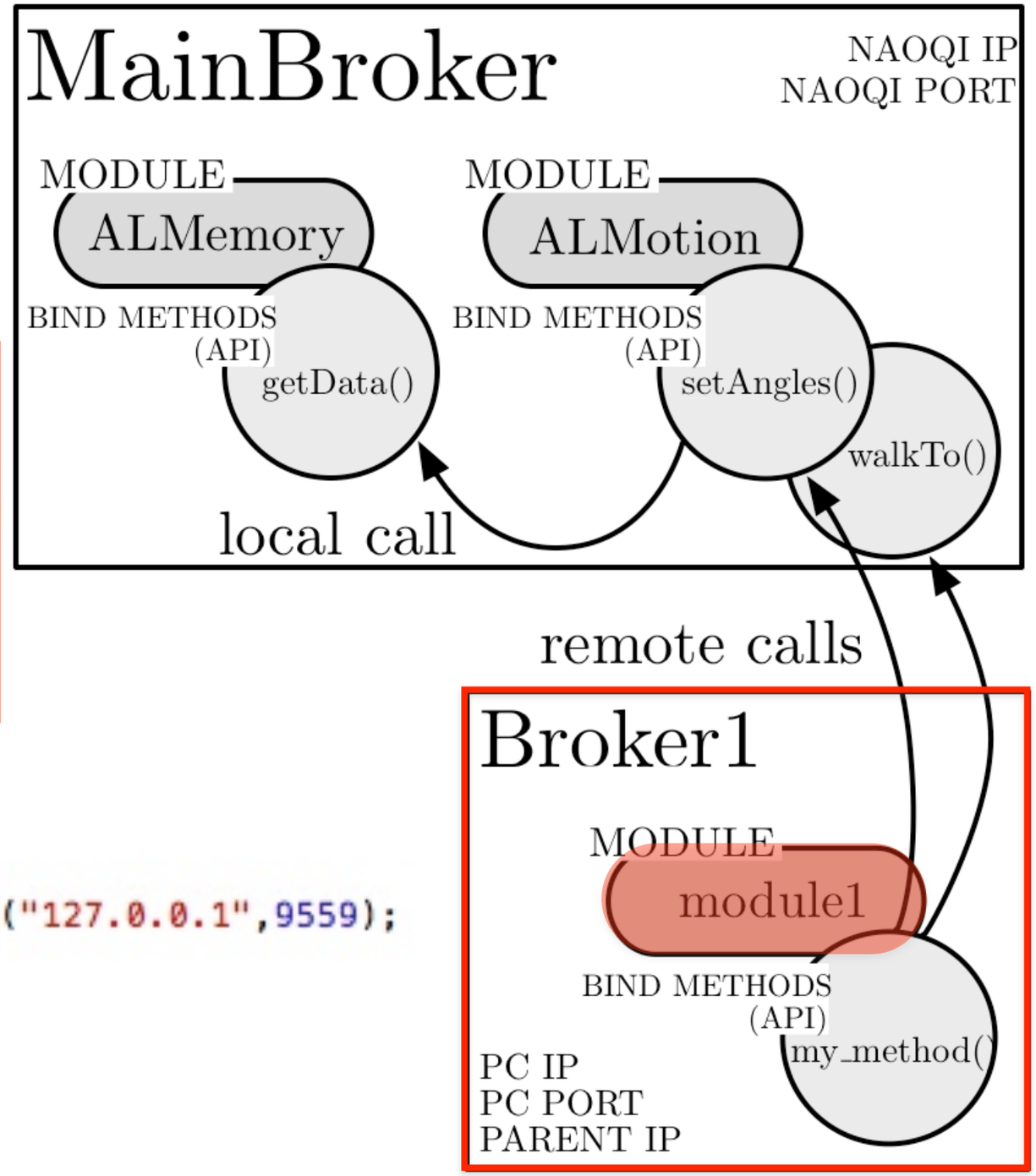


creation of an executable which make the robot walking and moves the yaw joint of the head

an example

```
ALCALL int _createModule( AL::ALPtr<AL::ALBroker> pBroker )
{
  // init broker with the main broker instance
  // from the parent executable
  AL::ALBrokerManager::setInstance(pBroker->fBrokerManager.lock());
  AL::ALBrokerManager::getInstance()->addBroker(pBroker);

  AL::ALModule::createModule<module1>( pBroker, "module1" );

  return 0;
}
```

```
void module1::my_method(){

  AL::ALMotionProxy *motion = new AL::ALMotionProxy("127.0.0.1",9559);

  motion->walkTo(0.5,0.0,0.0);

  motion->setAngles ("HeadYaw",0.4,0.2);

}
```

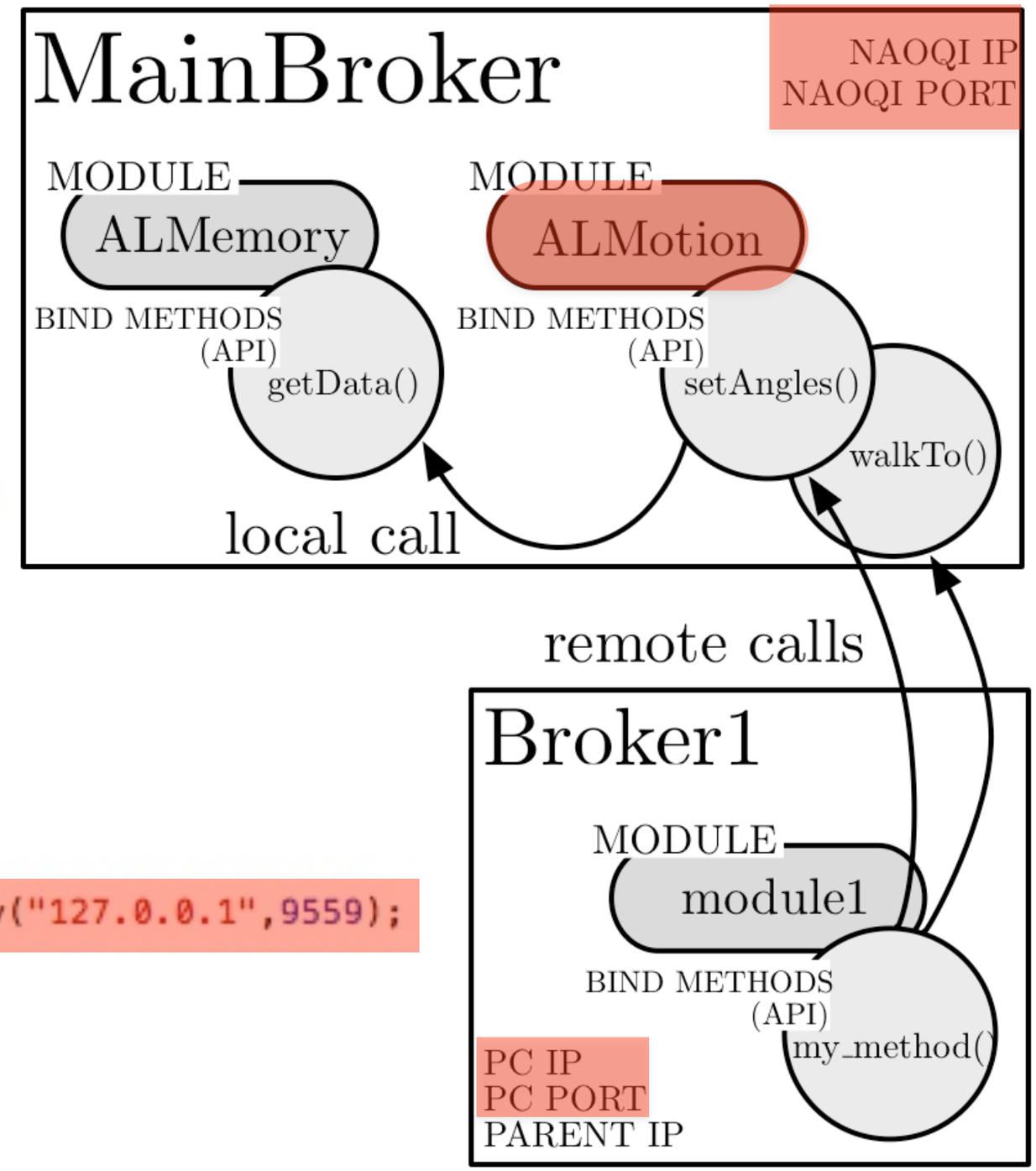


1. instantiation of the new module in a new broker created as an instance of the main broker

an example

```
ALCALL int _createModule( AL::ALPtr<AL::ALBroker> pBroker )
{
  // init broker with the main broker instance
  // from the parent executable
  AL::ALBrokerManager::setInstance(pBroker->fBrokerManager.lock());
  AL::ALBrokerManager::getInstance()->addBroker(pBroker);

  AL::ALModule::createModule<module1>( pBroker, "module1" );

  return 0;
}
```

```
void module1::my_method(){
  AL::ALMotionProxy *motion = new AL::ALMotionProxy("127.0.0.1",9559);
  motion->walkTo(0.5,0.0,0.0);
  motion->setAngles ("HeadYaw",0.4,0.2);
}
```

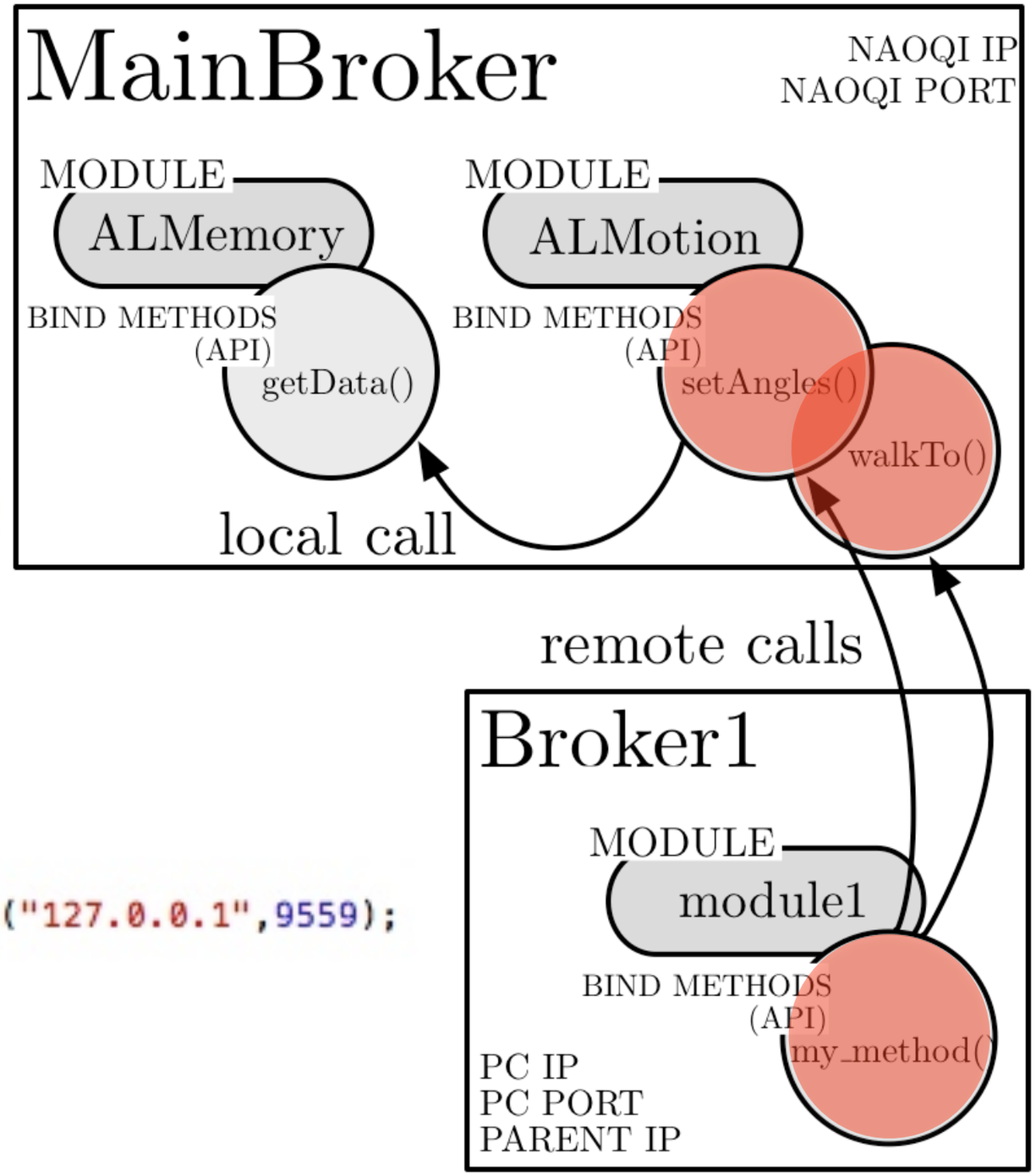


2. instantiation of a proxy to ALMotion module.

an example

```
ALCALL int _createModule( AL::ALPtr<AL::ALBroker> pBroker )
{
  // init broker with the main broker instance
  // from the parent executable
  AL::ALBrokerManager::setInstance(pBroker->fBrokerManager.lock());
  AL::ALBrokerManager::getInstance()->addBroker(pBroker);

  AL::ALModule::createModule<module1>( pBroker, "module1" );

  return 0;
}
```

```
void module1::my_method(){

  AL::ALMotionProxy *motion = new AL::ALMotionProxy("127.0.0.1",9559);

  motion->walkTo(0.5,0.0,0.0);

  motion->setAngles ("HeadYaw",0.4,0.2);

}
```

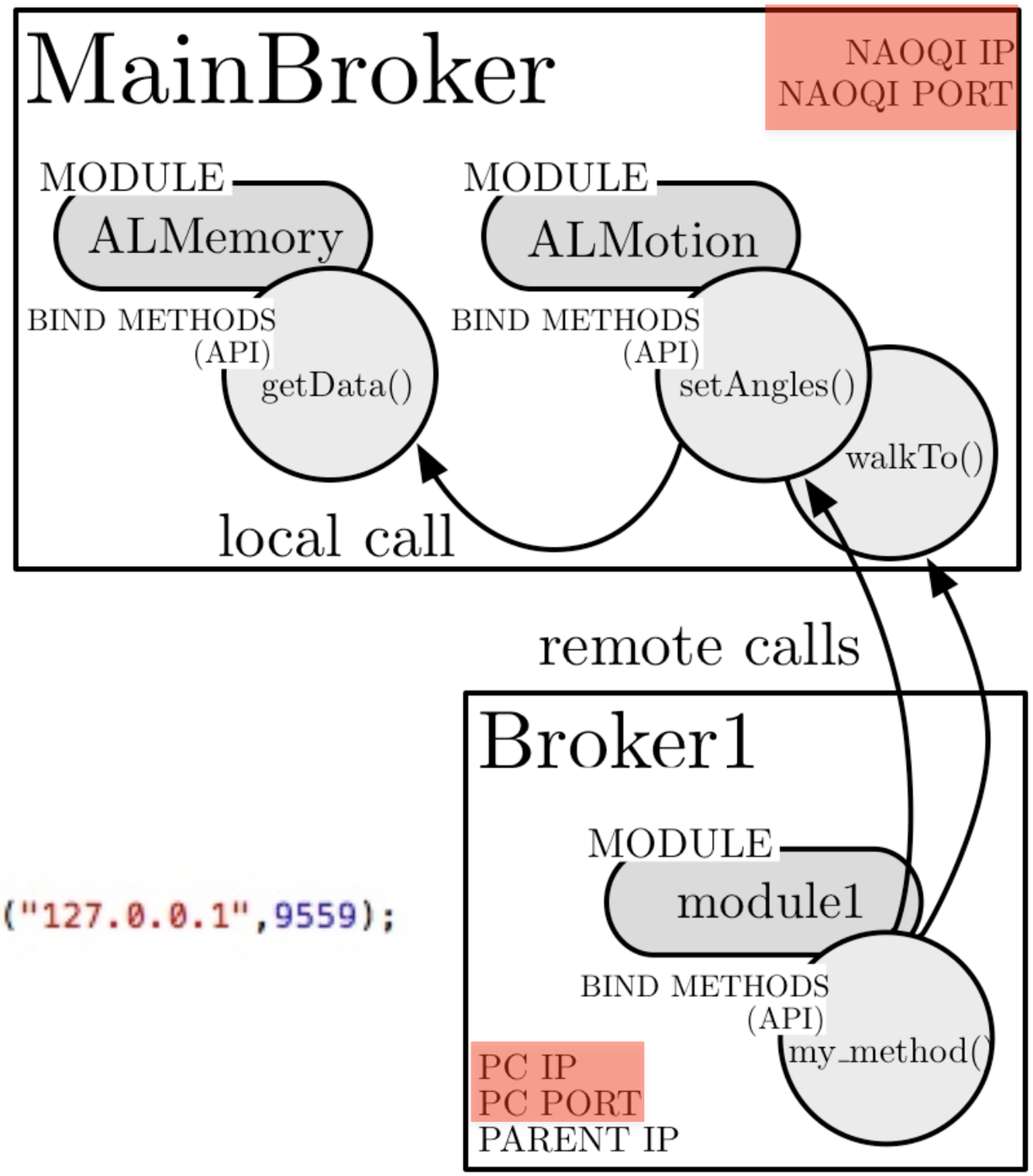


3. remote calls to ALMotion API methods

an example

```
ALCALL int _createModule( AL::ALPtr<AL::ALBroker> pBroker )
{
  // init broker with the main broker instance
  // from the parent executable
  AL::ALBrokerManager::setInstance(pBroker->fBrokerManager.lock());
  AL::ALBrokerManager::getInstance()->addBroker(pBroker);

  AL::ALModule::createModule<module1>( pBroker, "module1" );

  return 0;
}
```

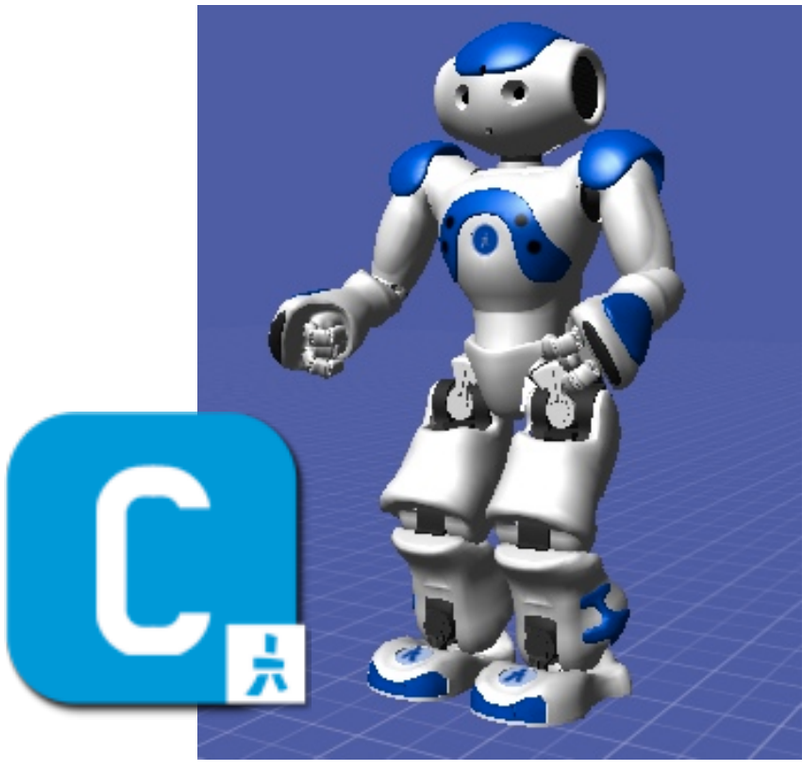
```
void module1::my_method(){

  AL::ALMotionProxy *motion = new AL::ALMotionProxy("127.0.0.1",9559);

  motion->walkTo(0.5,0.0,0.0);

  motion->setAngles ("HeadYaw",0.4,0.2);

}
```



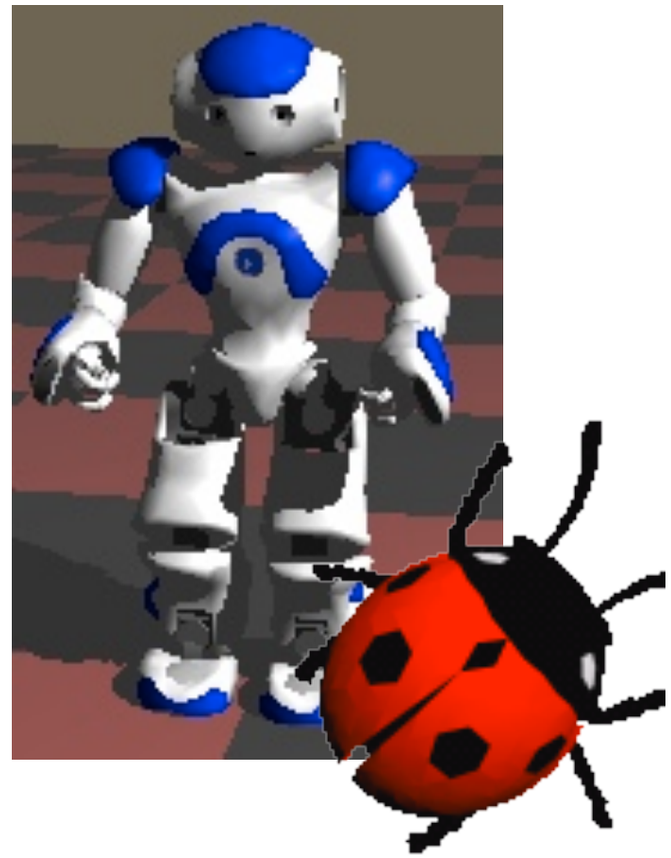
4. launch the executable

```
./module --pip <IP> --port <PORT>
```

how to see the output



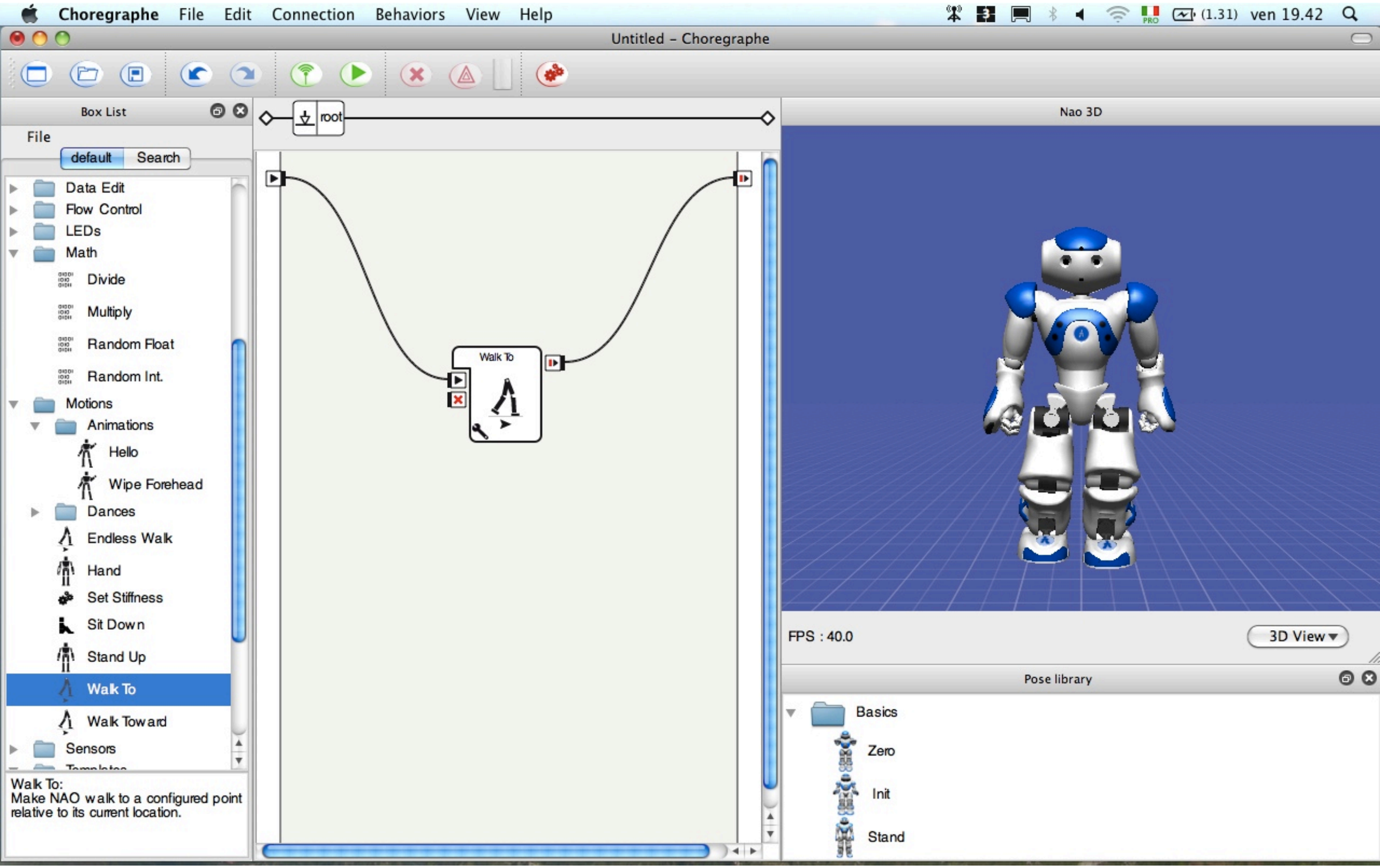
choregraphe



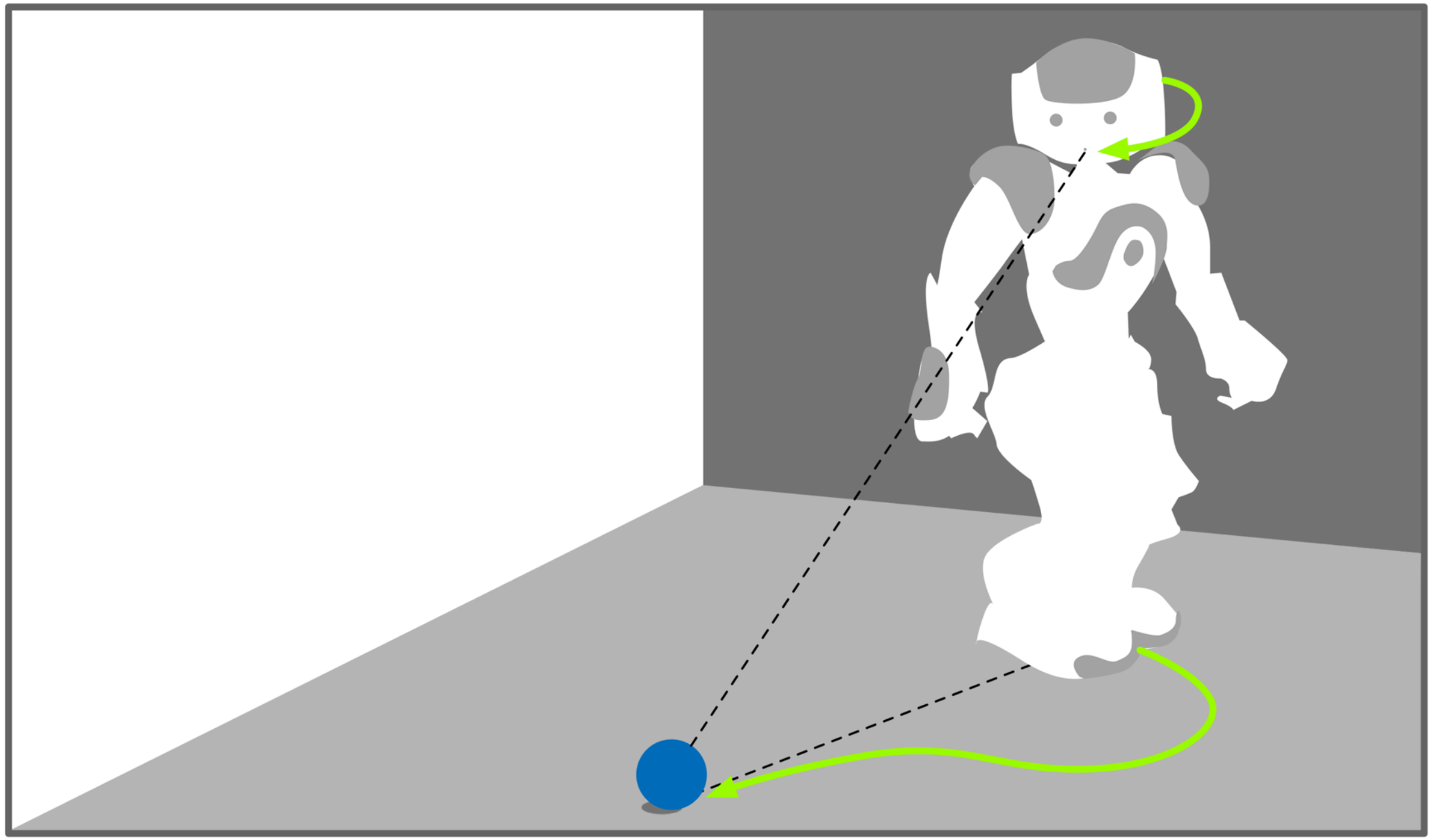
webots

real robot

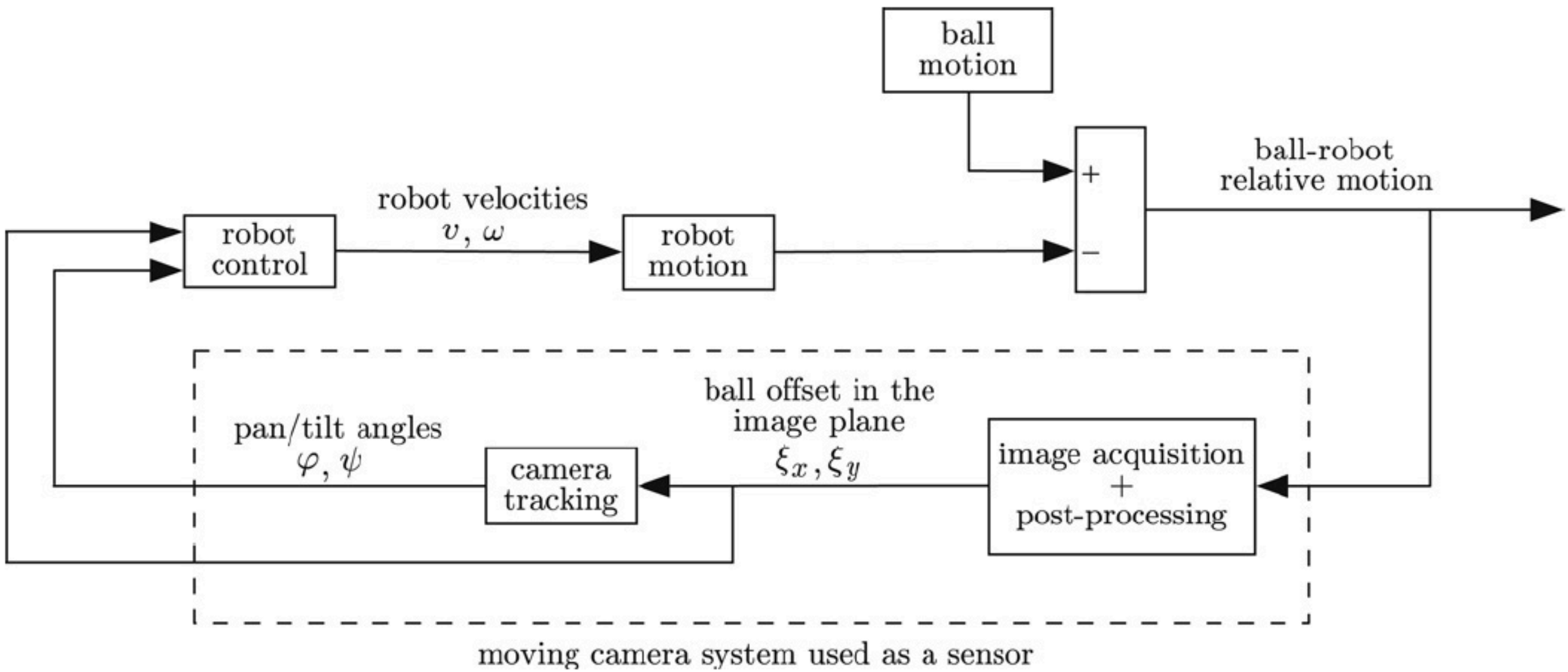
choregraphe



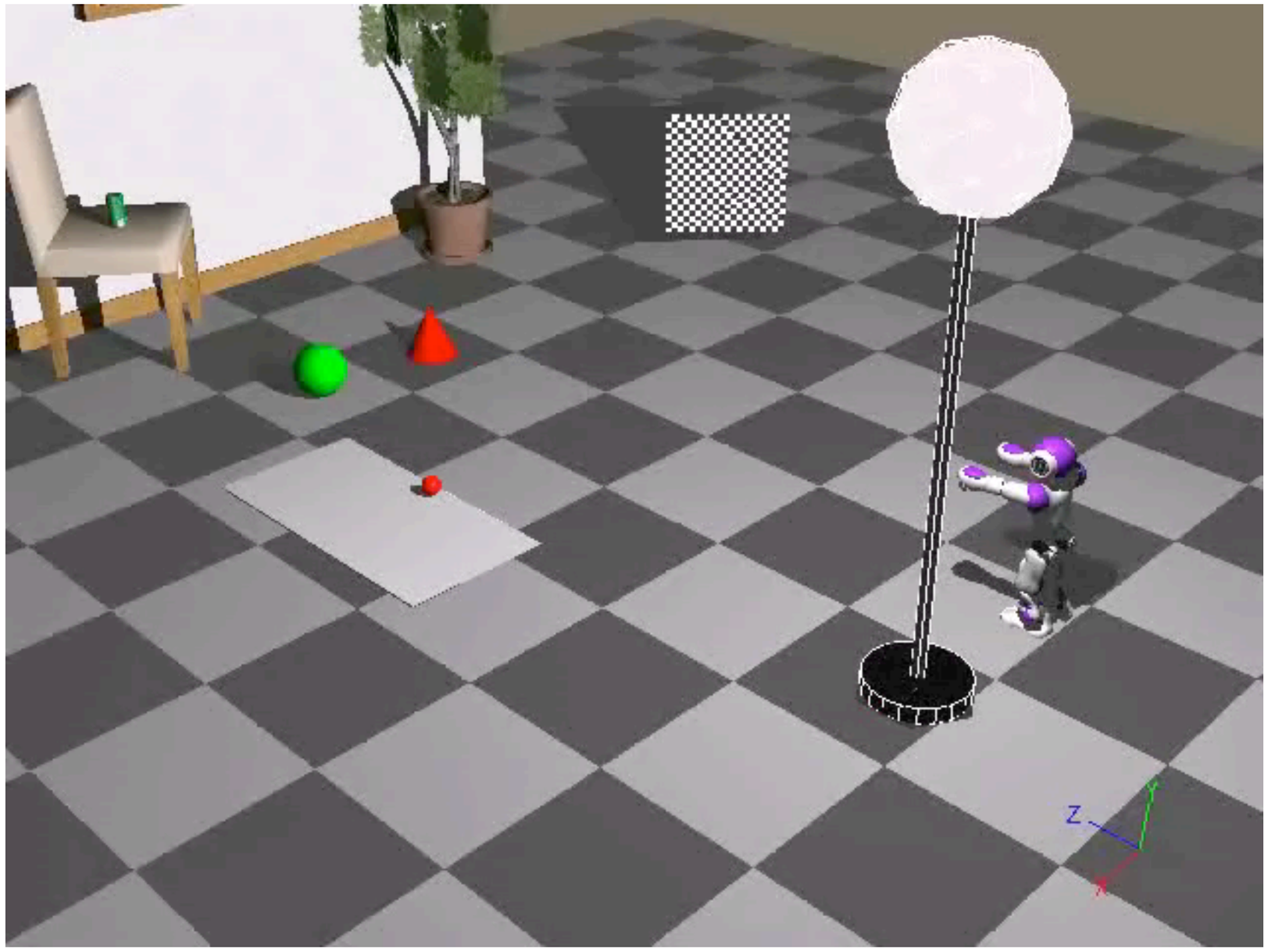
ball tracking



ball tracking



ball tracking



ball tracking