THE BIOROBOTICS INSTITUTE

Scuola Superiore Sant'Anna

Hands-On Session: Robotic Control

Robotics
M.Sc. programme in Computer Science

Lorenzo Vannucci
lorenzo.vannucci@santannapisa.it

Ugo Albanese
ugo.albanese@santannapisa.it

Alessandro Ambrosano
alessandro.ambrosano@santannapisa.it

March 15th, 2017

Goal of this hands-on session

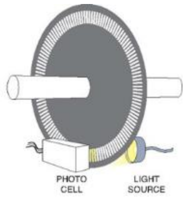
1. How robots can be controlled
2. Robotic simulators
3. Implementation of a low level controller

Goal of this hands-on session

1. **How robots can be controlled**
2. Robotic simulators
3. Implementation of a low level controller

Controlling robots

Sensors



Control Algorithm

```
while (true) {
  read_sensors()
  compute_motion()
  actuate()
  wait()*
}
```

Actuators

* loop should run at a fixed frequency

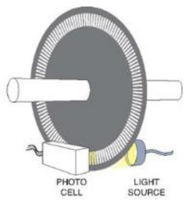
Controlling robots

Control Algorithm

```
read_sensors() {
  read_adc()
  count_steps()
  convert_to_angle()
}

actuate() {
  convert_angle_to_C()
  write_dac()
}
```

Sensors



Actuators

Controlling robots (today)

Control Algorithm

```
while (true) {
  receive_data()
  compute_motion()
  send_commands()
  wait()
}
```

actual joint angles, velocities, torques

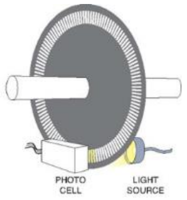
desired joint angles, velocities, torques

Robotics middleware

`read_sensors()`

`actuate()`

Sensors



Actuators

Robotics Middlewares

There is no standard solution, and in fact, there are many middlewares.

However, most of them provide at least:

- hardware abstraction
- inter-process communication

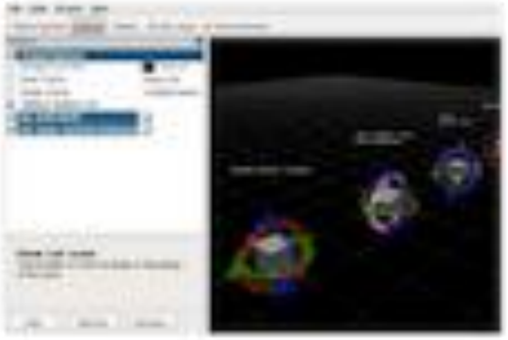
ROS

Orca

MIDDLEWARE

ROS: Robot Operating System

ROS is a **distributed**, **multi-lingual**, **open-source** robotic middleware.



Plumbing

- Process management
- Inter-process communication
- Device drivers

Tools

- Simulation
- Visualization
- Graphical user interface
- Data logging

Capabilities

- Control
- Planning
- Perception
- Mapping
- Manipulation

Ecosystem

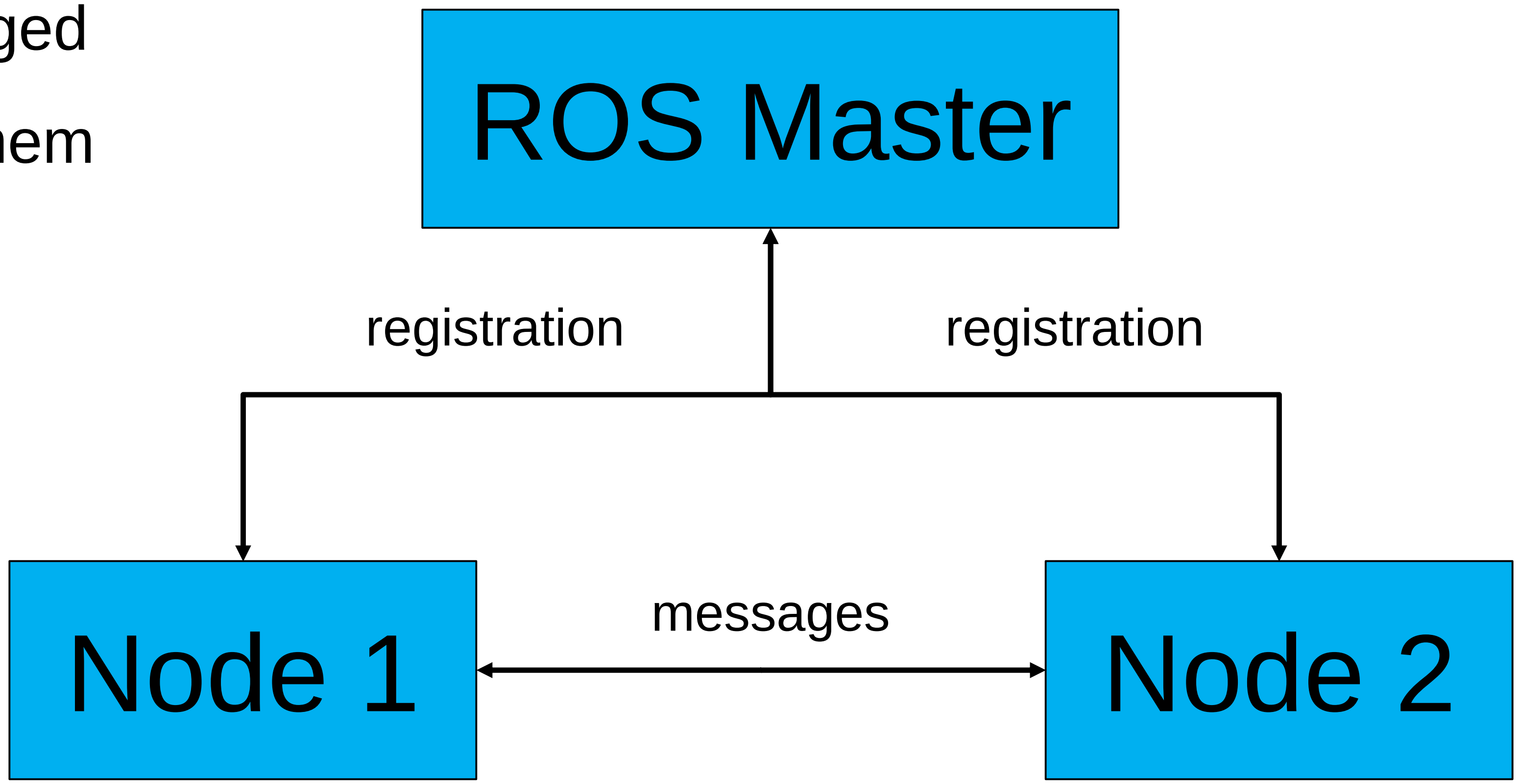
- Package organization
- Software distribution
- Documentation
- Tutorials

ROS Nodes

The **node** is the minimal execution element of a ROS architecture:

- single-purpose, executable program
- individually executed and managed
- exchange messages between them

The **ROS Master** manages the communication between nodes and every node registers at startup with the master.

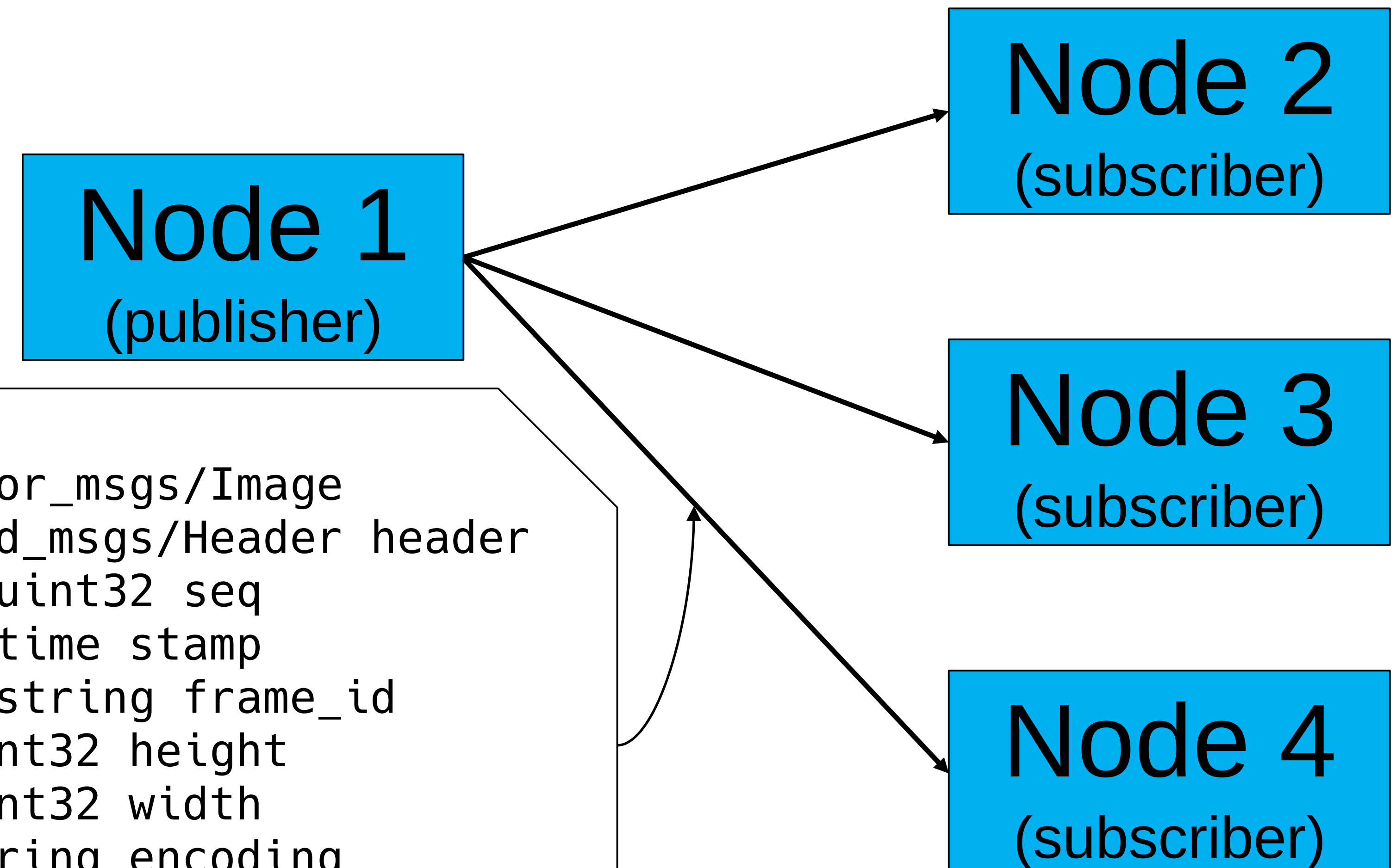


Nodes have two way to communicate between them: **topics** and **services**.

ROS Topics

ROS topics work using the **publish-subscribe** pattern:

- one publisher for every topic
- multiple subscribers
- single direction
- messages are typed
- asynchronous



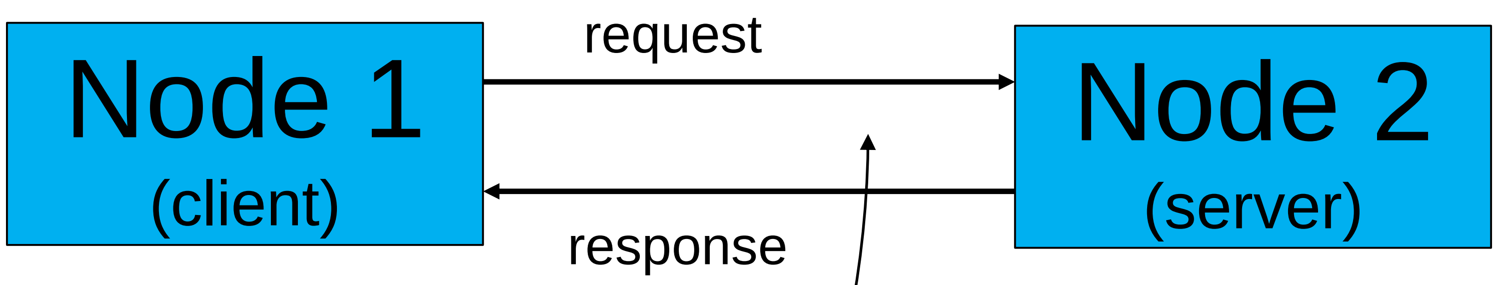

```
sensor_msgs/Image
  std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
  uint32 height
  uint32 width
  string encoding
  uint8 is_bigendian
  uint32 step
  uint8[] data
```

Let’s try it out!

ROS Services

ROS services are **remote procedure calls**:

- one node provides a service
- multiple nodes can use that service
- information is bidirectional
- messages are typed
- synchronous

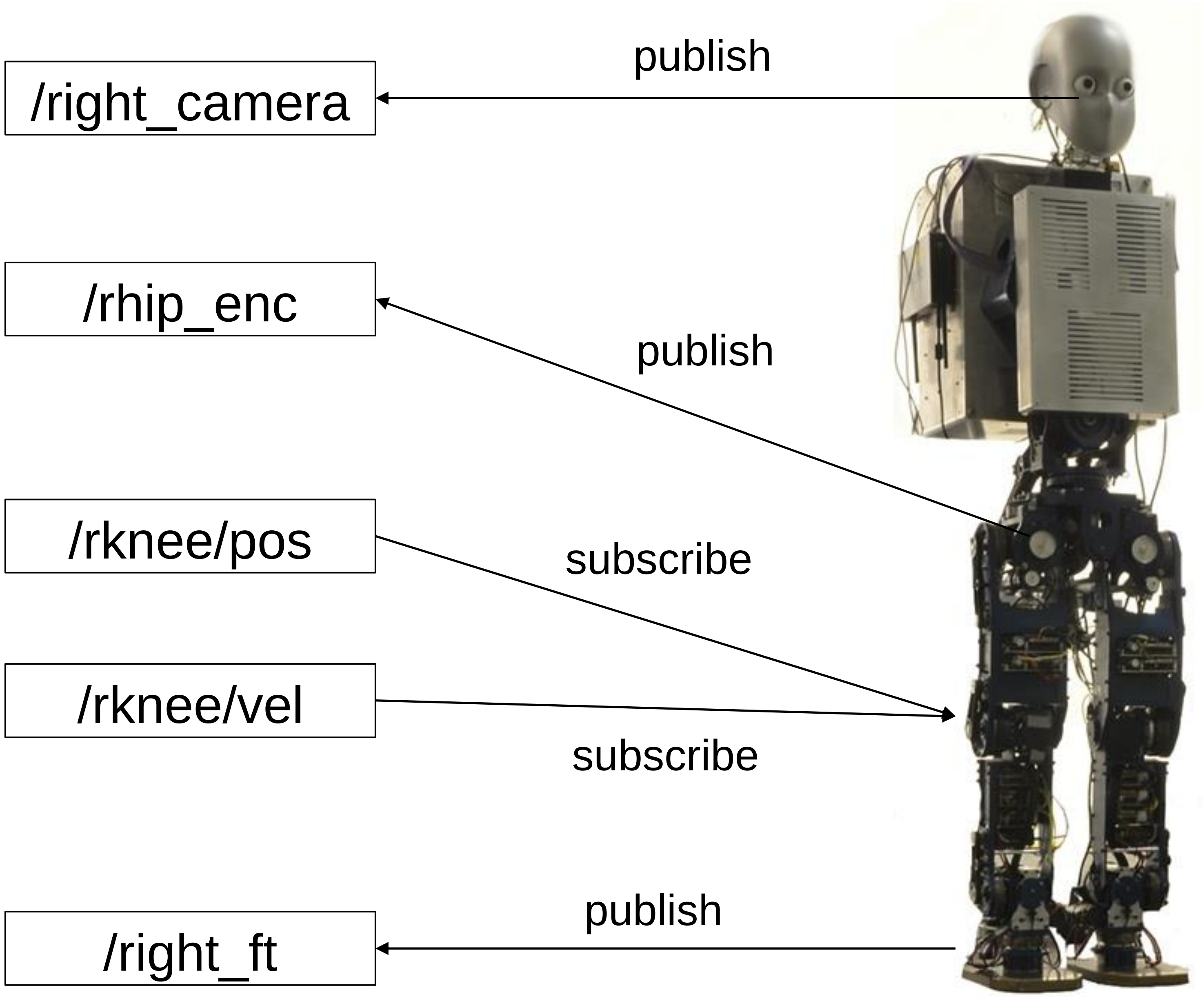



```
controller/SetPIDParameters
  string joint
  float64 kp
  float64 ki
  float64 kd
  ---
  bool success
```

Let’s try it out!

ROS Enabled Robots

A robot that publishes sensors reading an listens for motor commands by subscribing.



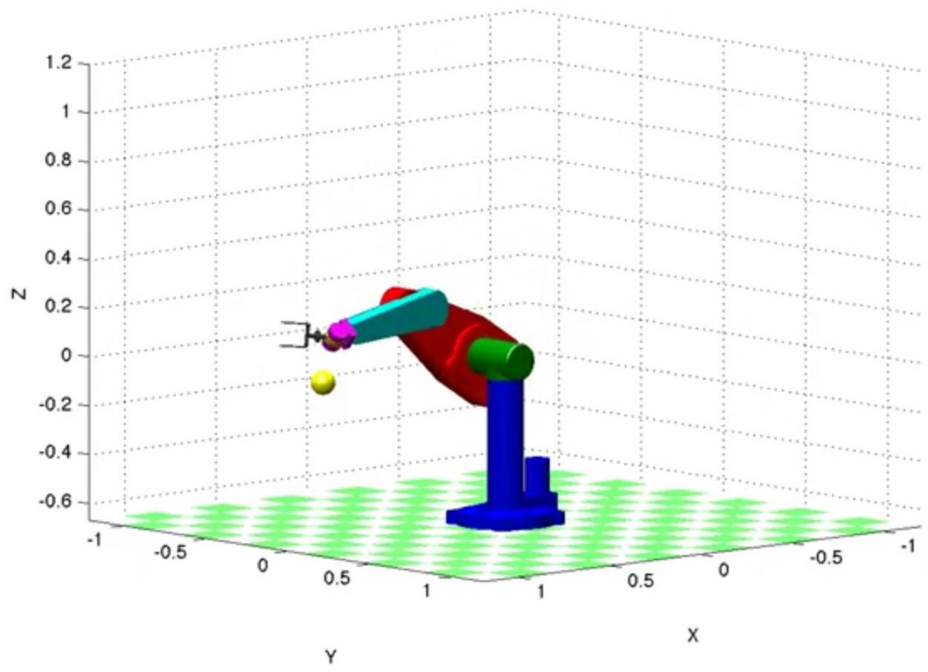
Goal of this hands-on session

1. How robots can be controlled
2. **Robotic simulators**
3. Implementation of a low level controller

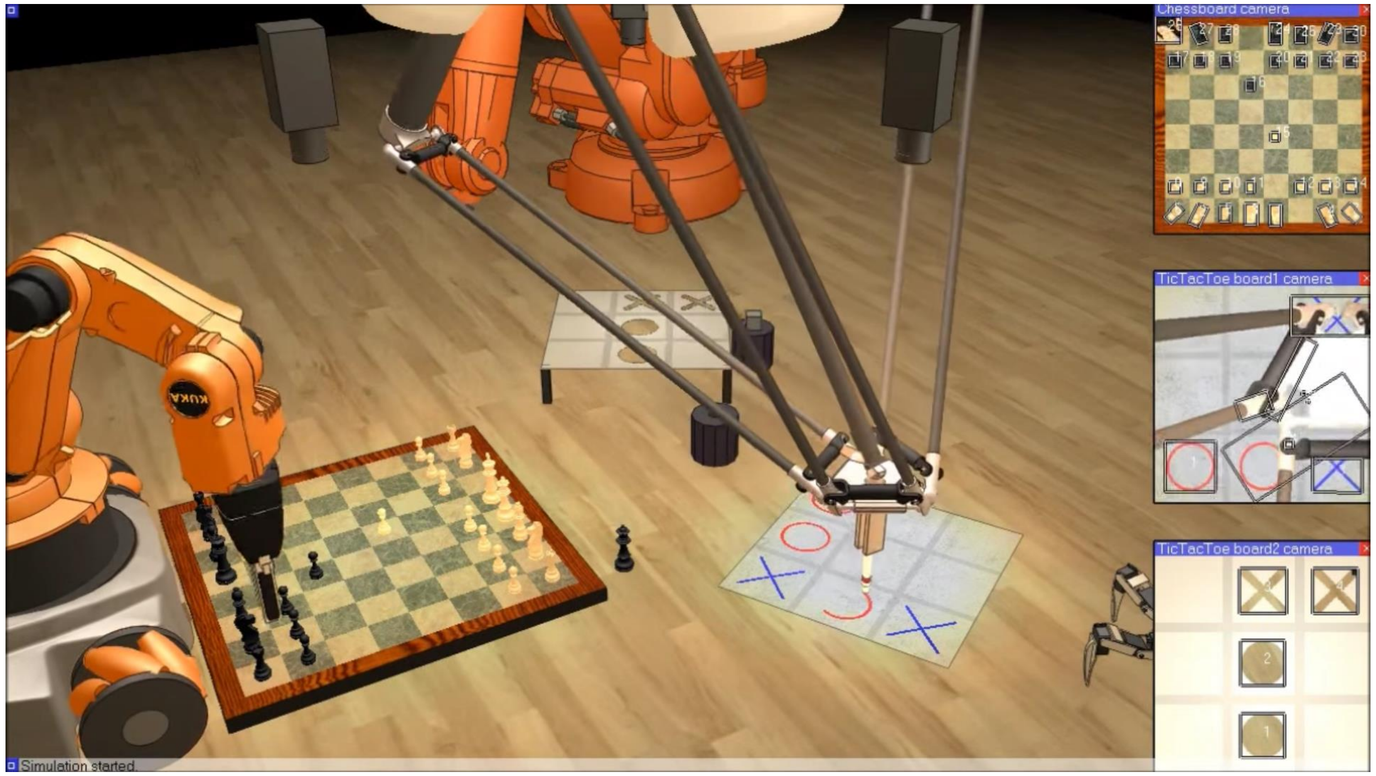
Robotic simulators

A robotic simulator is a piece of software mimics the robotic system and its surrounding environment to some level of accuracy. It may include:

- physics engine for more realistic simulations
- 3d rendering
- support for a robotic middleware



Robotics Toolbox



V-REP

Why simulate robots?

- cheaper: robots are expensive and may not be available
- rapid prototyping (sw): much faster to test control algorithms
- rapid prototyping (hw): much faster to test design changes

Why simulate robots?

- cheaper: robots are expensive and may not be available
- rapid prototyping (sw): much faster to test control algorithms
- rapid prototyping (hw): much faster to test design changes
- avoid these kind of situations →

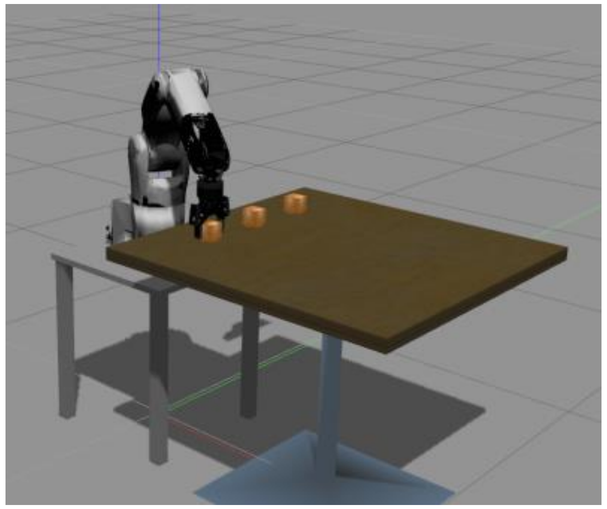
<u>However, it is important to remember that simulations are just simplified versions of reality, thus controllers that work in simulation may not work on the physical robot!</u>

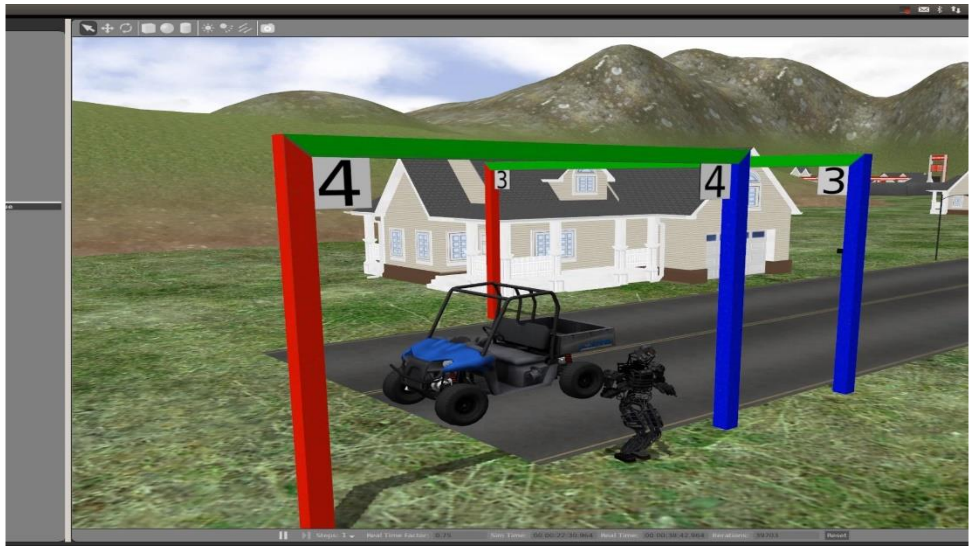
Gazebo

Gazebo is a robotic simulator (originally) developed as part of ROS.

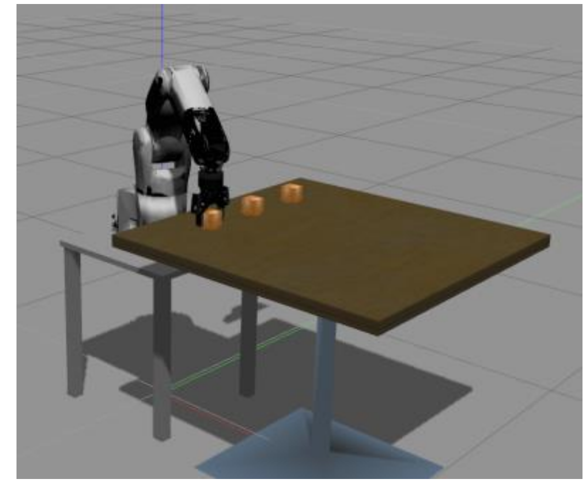
Main features:

- multiple physics engines supported (ODE, Bullet, Simbody, DART)
- 3d rendering via OGRE
- extensible through plugins (i.e. to add ROS/robots support)
- support for standard robot description formats (URDF, SDF)
- open source
- client-server architecture

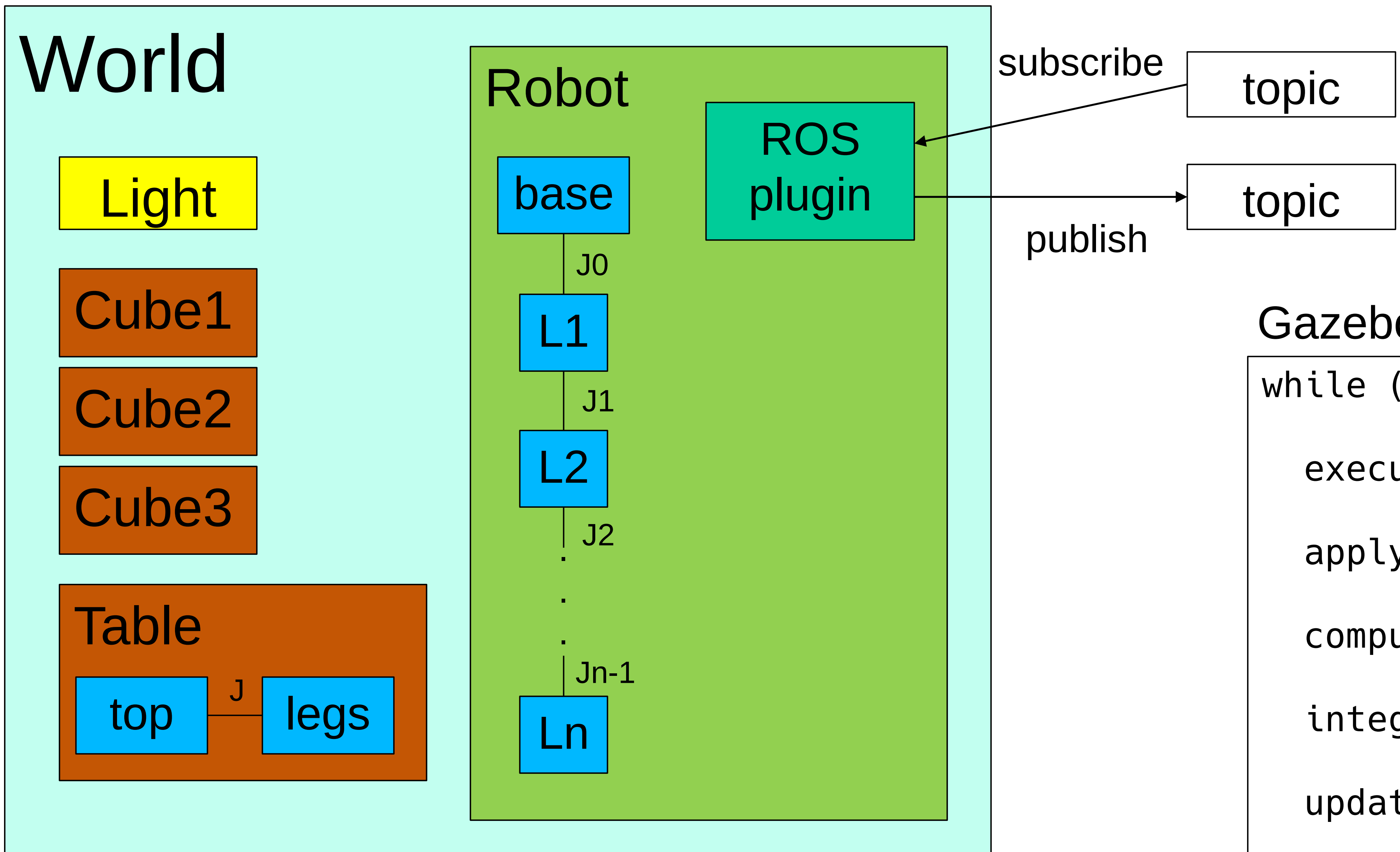




Gazebo



Inside a Gazebo simulation:

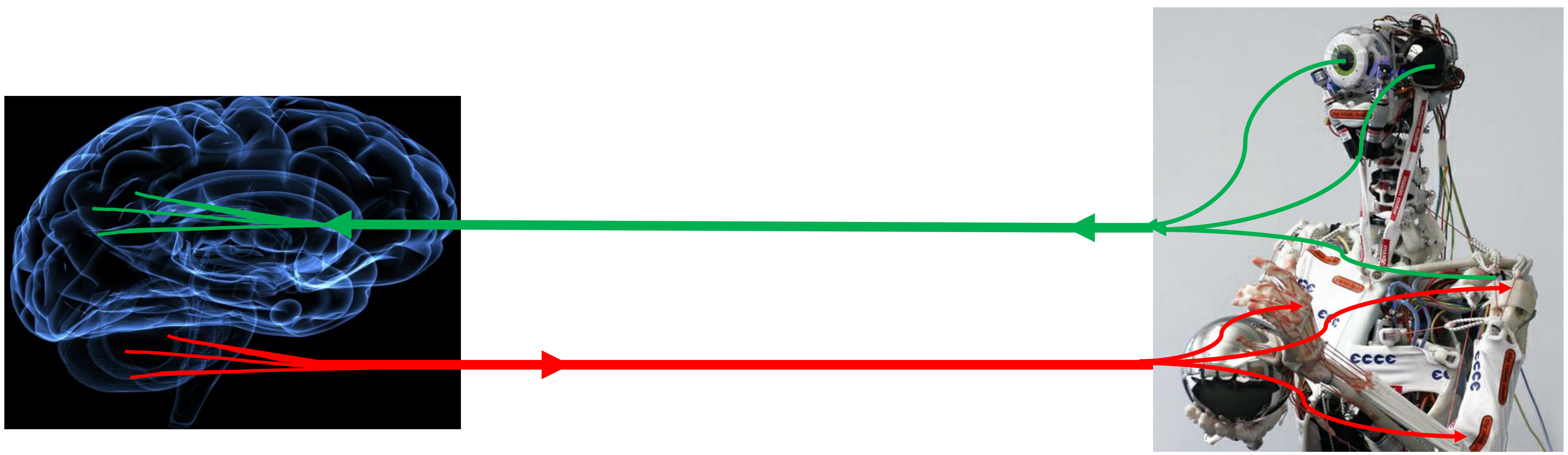


Gazebo simulation loop:

```
while (true) {
  execute_plugins()
  apply_forces_bodies()
  compute_collisions()
  integration_step()
  update_visuals()
}
```

The Neurorobotics Platform (NRP)

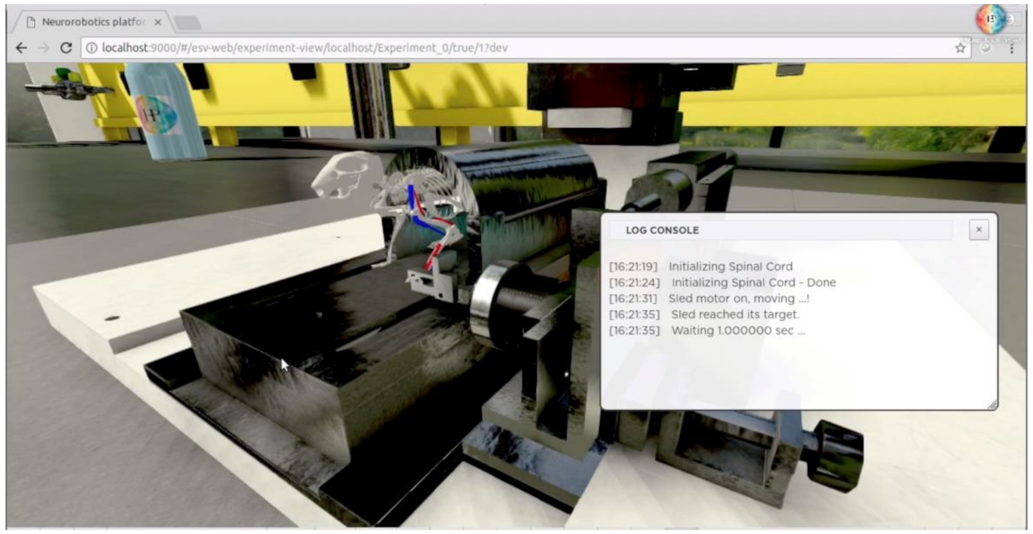
The Neurorobotics Platform is a simulation toolkit that aims at providing synchronized neural and robotic simulations, and data transfer from robot sensors/actors to brain areas and vice versa.



More on these kind of control methods in future lessons…

NRP features

- physical and robotic simulations are provided by Gazebo, via the ROS middleware
- web-based frontend for visualization and environment creation (replacing the standard Gazebo client)
- control loop implemented though a set of (transfer) functions that are called at every iteration of the loop
- includes a robot and environment designer
- more features will be discussed in future lessons

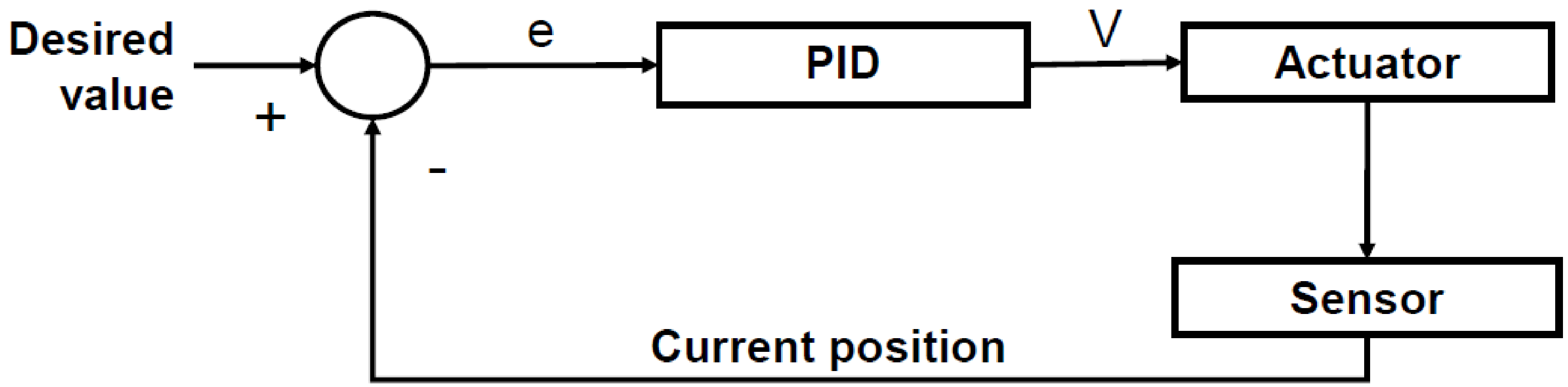


Goal of this hands-on session

1. How robots can be controlled
2. Robotic simulators
3. **Implementation of a low level controller**

PID controller

A PID controller is a low level feedback controller for a single joint capable of moving the joint from the current position to a desired position. This is done by converting the desired joint angle into an actuation signal (usually voltage).



$$V(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \dot{e}(t)$$

$$e(t) = q_d(t) - q_a(t)$$

PID controller implementation

Let’s try to do implement a discrete PID controller for the elbow joint of the iCub robot simulated in the NRP:

- simulation loop runs at 50Hz (every 20ms of simulated time)
- control signal is the elbow joint velocity, not voltage
- target motion is $q_d(n) = 0.8 * \sin(t) + 1.1$
- discrete PID $V(n) = K_p e(n) + K_i e_i(n) + K_d e_d(n)$

$$e(n) = q_d(n) - q_a(n)$$

$$e_i(n) = e_i(n-1) + \frac{(e(n) - e(n-1))\Delta t}{2} \qquad e_d(n) = \frac{e(n) - e(n-1)}{\Delta t}$$

Hands-On Resources

ROS:
- www.ros.org

Gazebo:
- gazebosim.org

Neurorobotics Platform:
- neurorobotics.net

Others (related):
- lorenzo.vannucci@santannapisa.it
- ugo.albanese@santannapisa.it
- alessandro.ambrosano@santannapisa.it