THE BIOROBOTICS INSTITUTE

Scuola Superiore Sant'Anna

Neuromorphic computing

Robotics
M.Sc. programme in Computer Science

Lorenzo Vannucci
lorenzo.vannucci@santannapisa.it

April 26th, 2018

Outline

1. Introduction
2. Fundamentals of neuroscience
3. Simulating the brain
4. Software and hardware simulations
5. Robotic applications

Outline

1. Introduction
2. Fundamentals of neuroscience
3. Simulating the brain
4. **Software and hardware simulations**
5. Robotic applications

Point neuron simulators - NEST

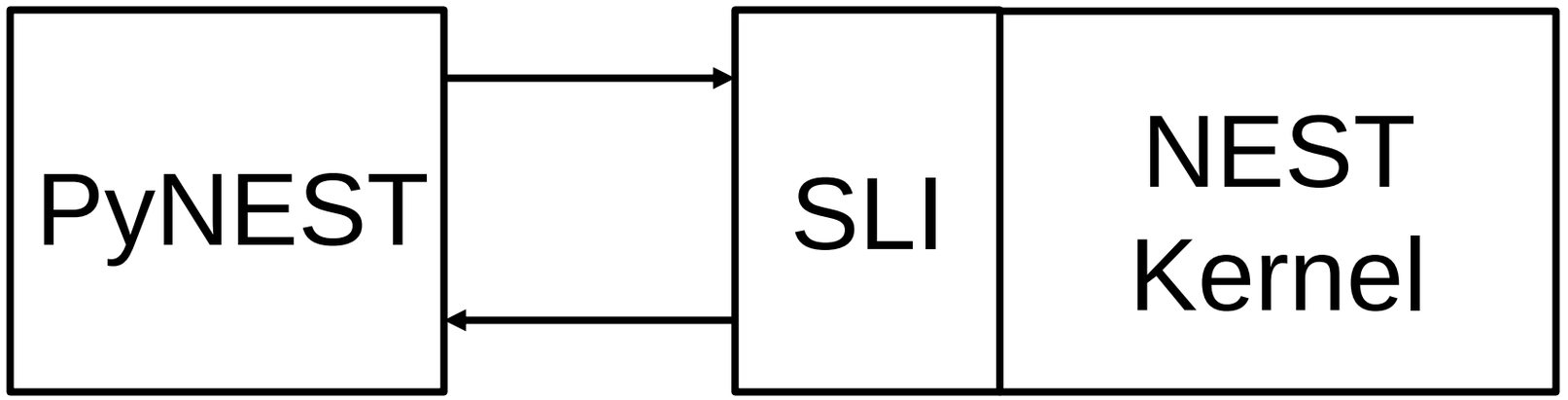
We don’t have to implement a whole simulator by ourselves, several already exists!

Among these, a popular choice is NEST (NEural Simulation Tool), an open source spiking neural network simulator developed by the NEST initiative (www.nest-simulator.org). Among its features, there are:

- over 50 neuron models (including LIAF, Hodgkin-Huxley and Izhikevich)
- over 10 synapse models (including STDP)
- minimal dependencies
- open source (GNU GPLv2)
- “easily” extendable

Point neuron simulators - NEST

NEST has a simulation kernel (written in C++) and two layers of interface towards it.



The kernel cannot be directly accessed. In fact, the executable launches the Simulation Language Interpreter to which one can send commands to create the network.

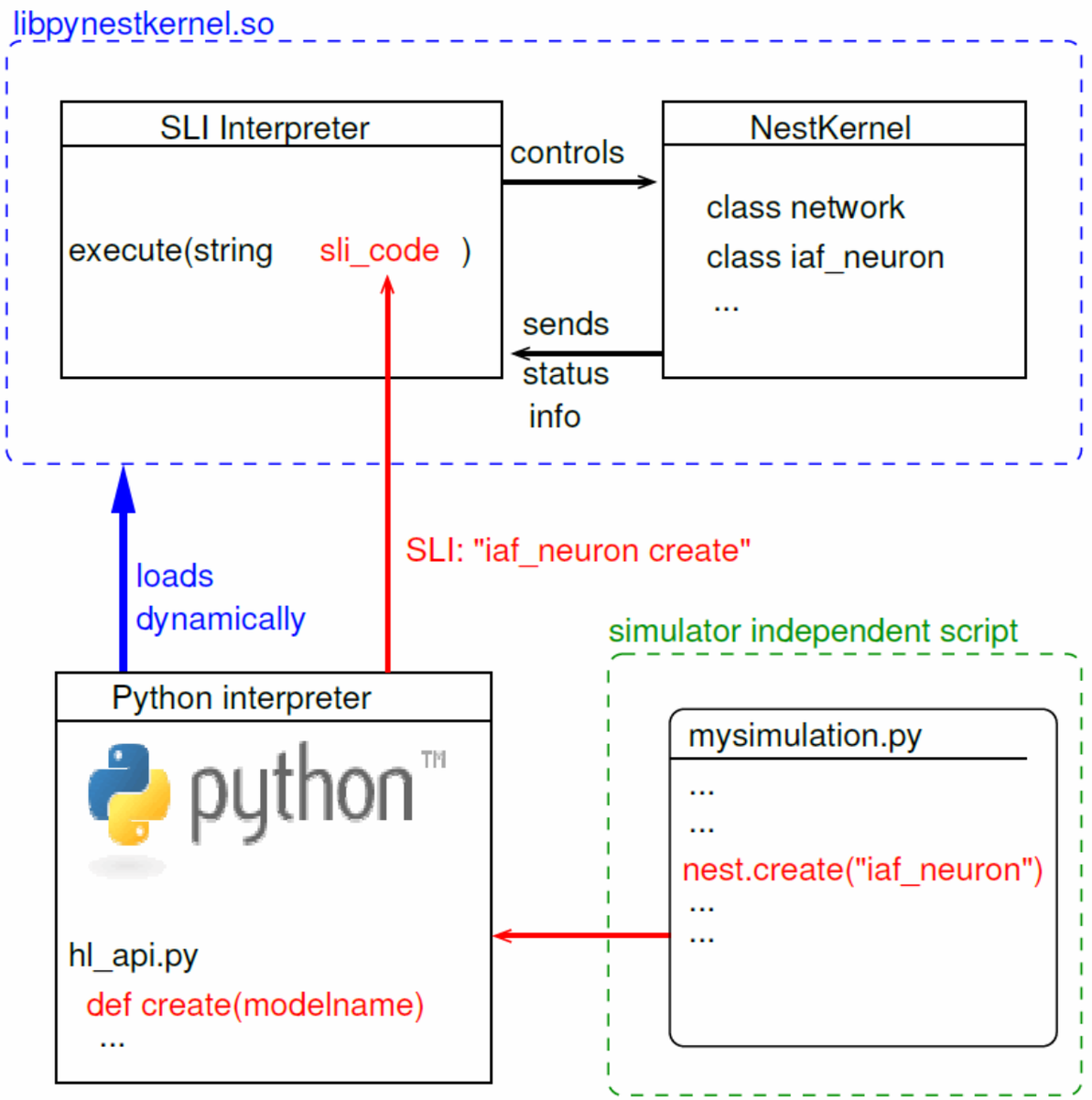
Why PyNEST? Because SLI is basically PostScript!

```
/iaf_neuron Create /n Set
/poisson_generator Create /pg Set
pg << /rate 220.0 Hz >> SetStatus
pg n Connect
```

```
n = nest.Create('iaf_neuron')
pg = nest.Create('poisson_generator')
nest.SetStatus(pg, {'rate': 220.0})
nest.Connect(pg, n)
```

Point neuron simulators - NEST

PyNEST provides an usable interface towards SLI.



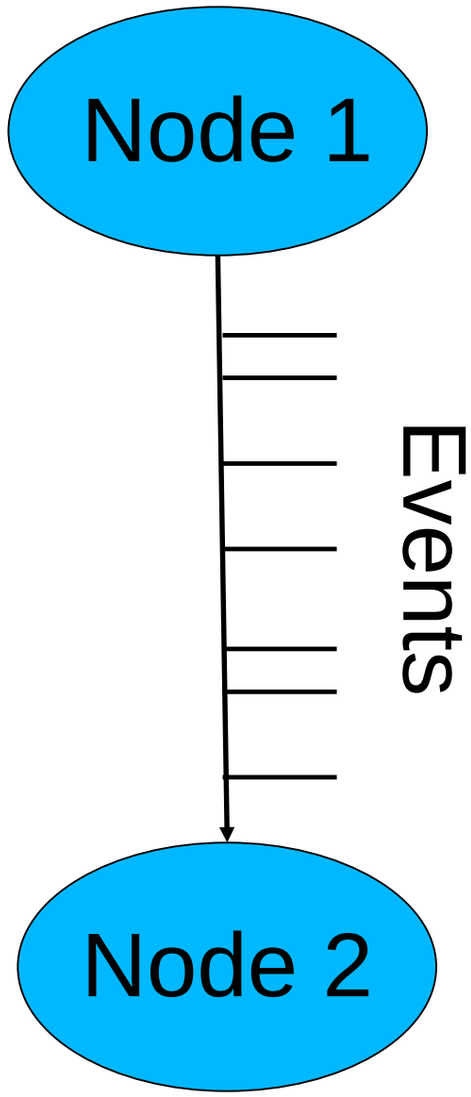
Point neuron simulators - NEST

A NEST network is a *directed weighted graph*:

- **Nodes**
  - neurons, devices, sub-networks
  - have a dynamic state that changes over time and can be influenced by or produce events

- **Events**
  - pieces of information of a particular type (e.g. spike, voltage or current event)

- **Connections**
  - communication channels for the exchange of events
  - directed (pre to post)
  - weighted (synaptic weights)
  - delayed (delay must be greater than 0!)



Point neuron simulators - NEST

The simulation is discretized into time steps of a certain duration (Δt). The simulation loop works as follows:

1. PSC for all delivered events are computed
2. membrane potential is updated and new events are bufferized
3. new events are sent towards post-synaptic nodes
4. simulation time is increased by Δt

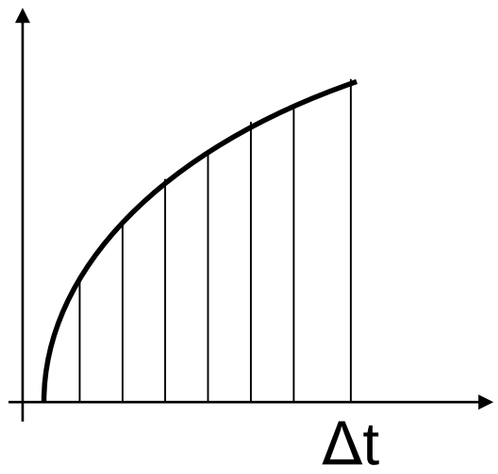
Point neuron simulators - NEST

The simulation is discretized into time steps of a certain duration (Δt). The simulation loop works as follows:

1. PSC for all delivered events are computed
2. membrane potential is updated and new events are bufferized
3. new events are sent towards post-synaptic nodes
4. simulation time is increased by Δt

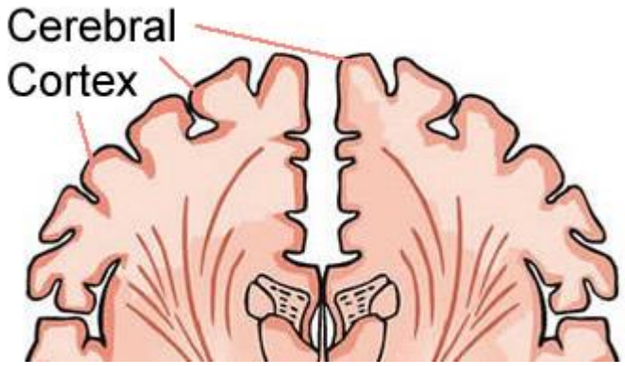
Notes:

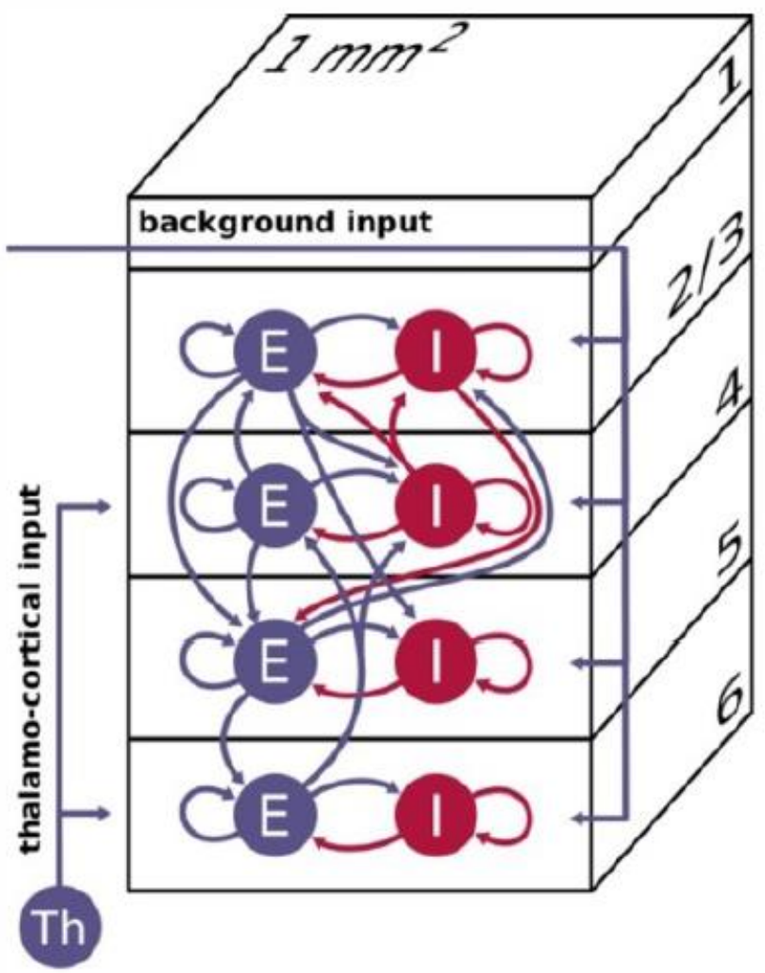
- actually 1 and 2 occur inside an inner loop with a time step < Δt!
- delay of connections must be >= Δt
- during the time step, the node is isolated from the rest



NEST examples – cortical microcircuit

We want to simulate (a layer of) the cerebral cortex:





- $1mm^2$
- 0.3 billion synapses, 80000 neurons
- 6 layers
- 2 population of LIAF neurons per layer

Potjans, Tobias C., and Diesmann, Markus. “The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model.” Cerebral Cortex 24.3 (2014): 785-806

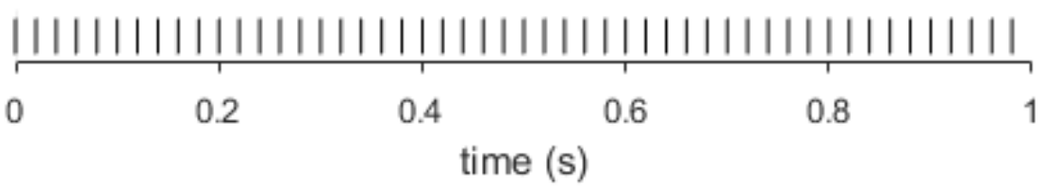
Let’s try it out!

NEST examples – Poisson generators

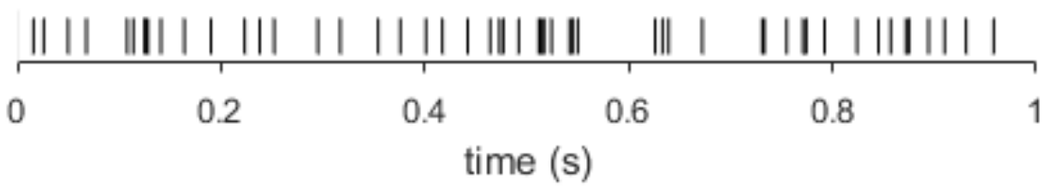
In order to give inputs to the system, representing activity of brain areas not modelled or sensory information, we need to **generate spikes** without actually simulate neurons.

It has been observed that most of the times (excluding when time encoding mechanisms are in action) the timing of successive action potential is highly irregular, probably because of stochastic forces.

Thus, when generating spikes, we want to avoid generating uniformly spaced action potentials.



Bad



Good

NEST examples – Poisson generators

To generate irregular spikes we can assume that every spike is independent from the previous one and that the generation depends solely on the instantaneous firing rate.

Upon this hypothesis we can generate spikes using a Poisson process:

$$P\{n\ spikes\ during\ \Delta t\} = e^{-r\Delta t}\frac{(r\Delta t)^n}{n!}$$

$$P\{1\ spike\ during\ \Delta t\} \approx r\Delta t\ ^*$$

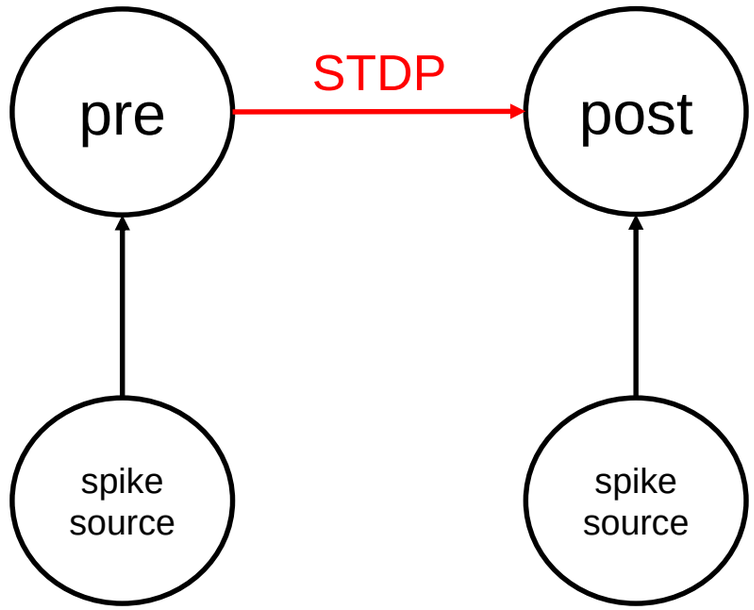
* For a sufficiently short Δt

NEST examples – STDP

Let’s see an example of synaptic plasticity.

- two populations, connected with STDP-enabled synapses
- external spike sources that trigger activity
- execution phases and expected results:

1. only pre stimulation → no post activity
2. pre and post stimulation → plasticity
3. only pre stimulation → also post activity



Let’s try it out!

NEST examples

What did we learn from these example?

- using nest is very easy to set up neural simulation
- nice syntactic sugar for randomized connection and weights
- useful spike recording utilities

- but it can take more than 10 seconds to simulate 1!

```
start_time = time.time()
nest.Simulate(T)
elapsed_time = time.time() - start_time
print elapsed_time
```

```
11.4397270679
```

We need to find a way to *speed up* the simulation.

NEST – parallel simulations

Let’s recall the kernel simulation loop:

1. PSP for all delivered events are computed
2. membrane potential is updated and new events are bufferized

MPI thread

3. new events are sent towards post-synaptic nodes
4. simulation time is increased by Δt

MPI process

Inside a single time step, each neuron is decoupled from the others, thus the simulation of a single time step is an embarrassingly parallel problem.

In fact, NEST natively supports MPI and the parallelization of the loop.

Moreover, MPI is supported on High Performance Computing platforms!

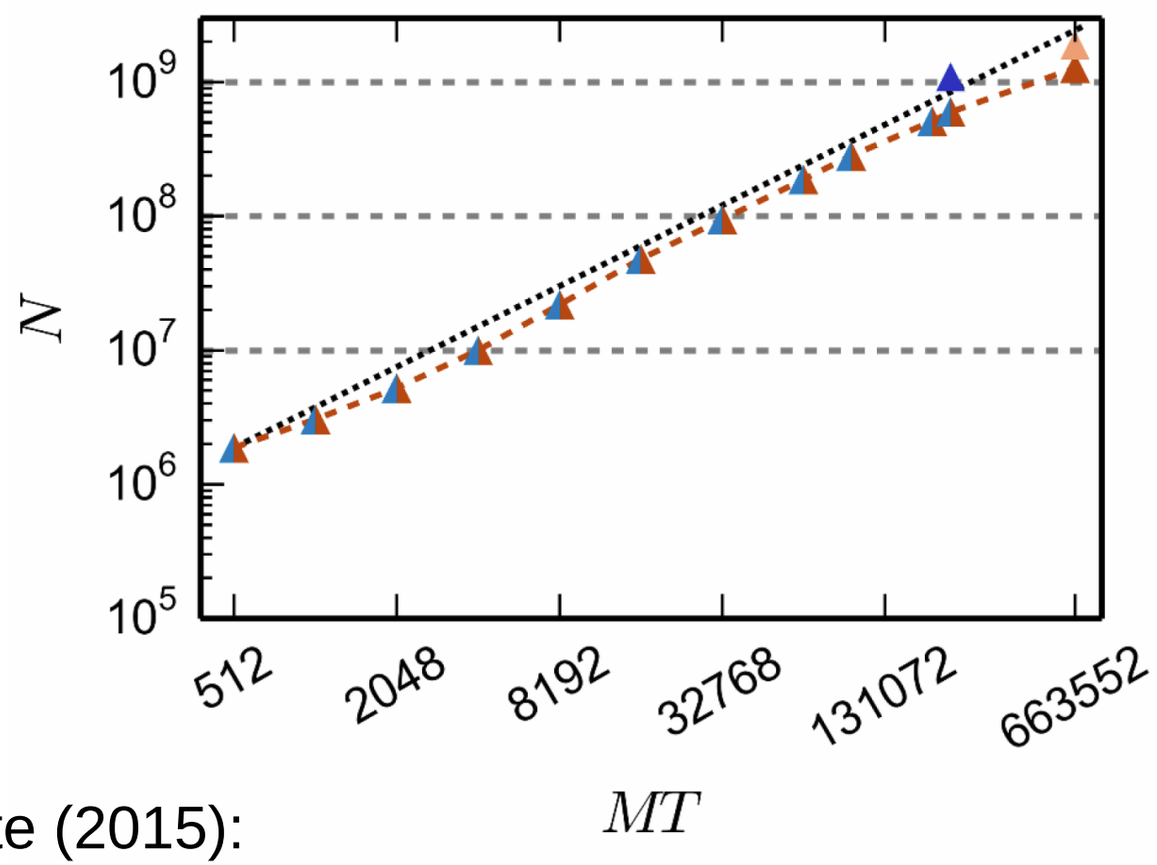
NEST – HPC

How well does nest perform on supercomputers?

Legend:

K - RIKEN, Japan
663,552 nodes, 4th

JUQUEEN - Jülich, Germany
229,376, 11th



Largest network simulation performed to date (2015):

1.86x10^9 neurons, 6000 synapses each

1.08x10^9 neurons, 6000 synapses each

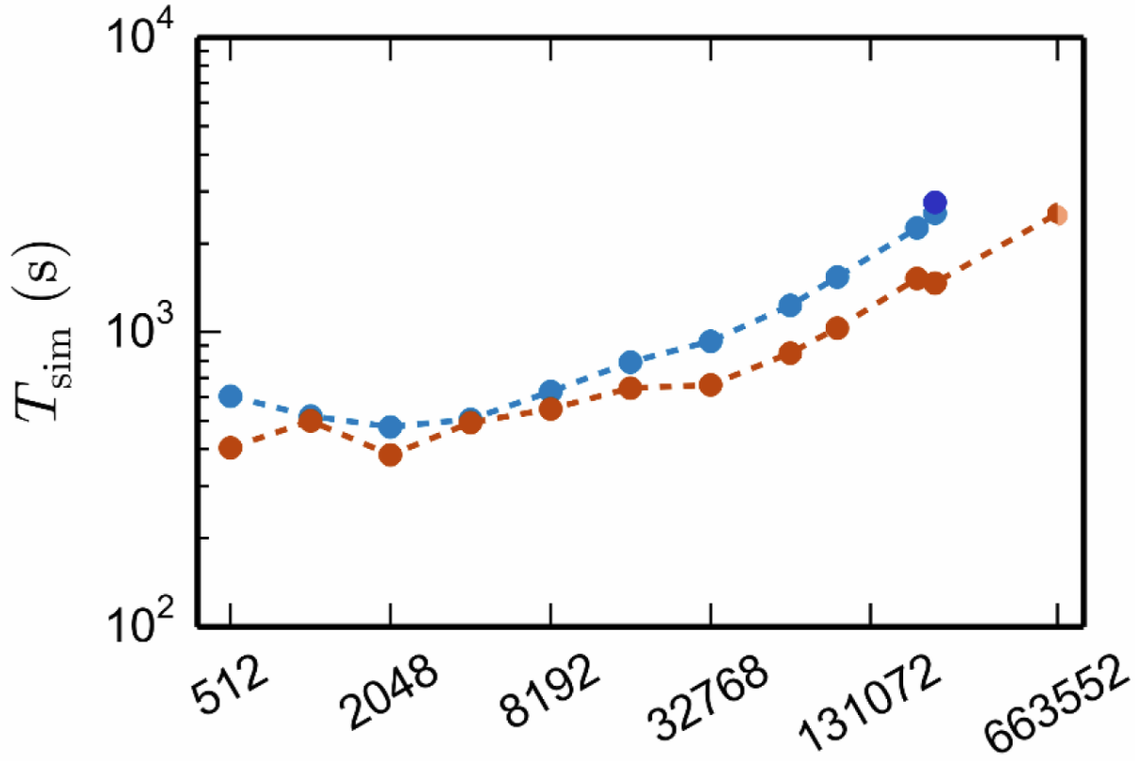
NEST – HPC

How well does nest perform on supercomputers?

Legend:

K - RIKEN, Japan
663,552 nodes, 4th

JUQUEEN - Jülich, Germany
229,376, 11th



Simulation time of 1 second of real time varies between:

between 6 and 42 minutes

between 8 and 41 minutes

NEST – Final Remarks

Is this a viable solution for physical robotics? Not really.

- even if we would like to simulate smaller networks, simulations will not be **real-time**
- usually robotics labs do not have supercomputers
- supercomputers work with job systems and as of today no interactive job mechanism exists
- latencies between the supercomputer and the robot
- power consumption (9.89 MW for K supercomp.)

However, NEST can be coupled with **robotics simulations** (more on this later).

Neuromorphic hardware

A new kind of processors, specifically designed to compute neural dynamics, have been developed in the last few years. This is what is called **neuromorphic hardware**.

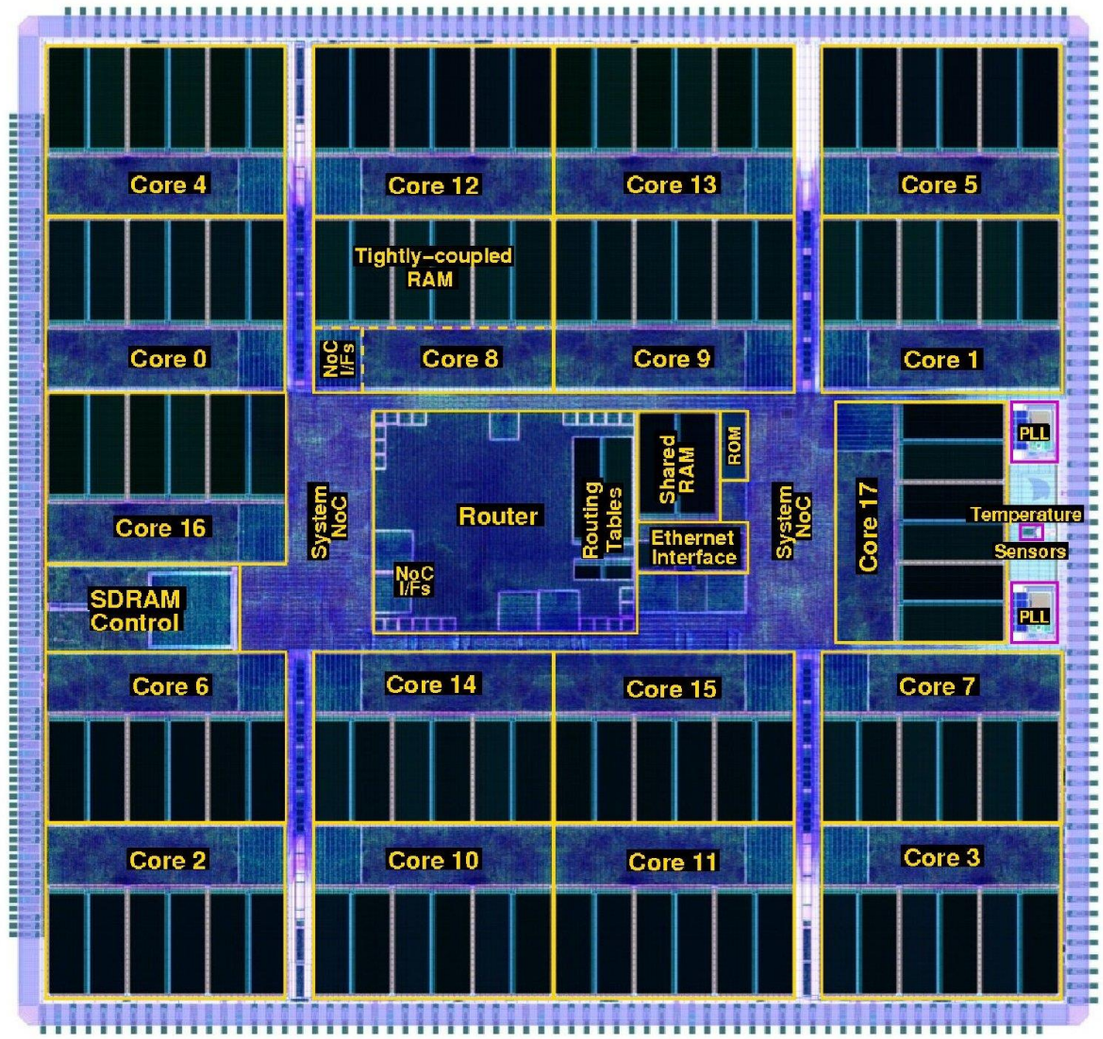
Usually, these kind of processors have these characteristics:

- massively parallel computation
- energy efficiency
- fault tolerance
- self organization of the network
- fast simulation times
- compactness

Neuromorphic hardware – SpiNNaker

SpiNNaker is a neuromorphic hardware platform developed by the University of Manchester.

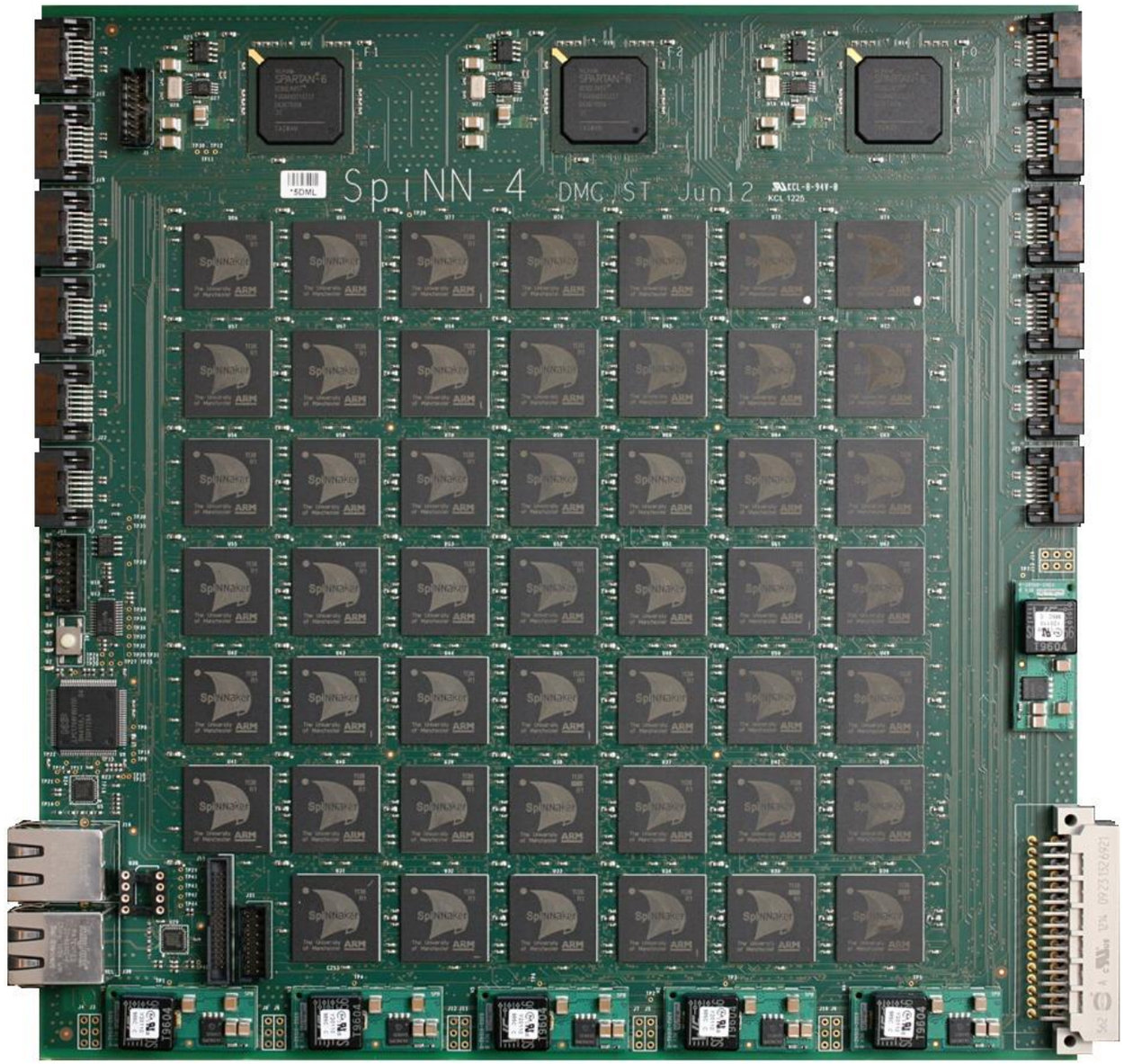
- 1W chip
- 18 ARM-968 cores
- 1Gbit DDR-2 SDRAM
- 240MHz
- 6 bi-directional links
- optimized for 10million 32-bit packets/s



Neuromorphic hardware – SpiNNaker

SpiNNaker cores are arranged on 48 chips boards.

- 1000 neurons per core
- 18000 neurons per chip
- 864000 neurons per board
- 3.1Gbps SATA connections for connecting to other boards
- two 100Mbps Ethernet for control
- max 70W consumption, low temp



Neuromorphic hardware – SpiNNaker

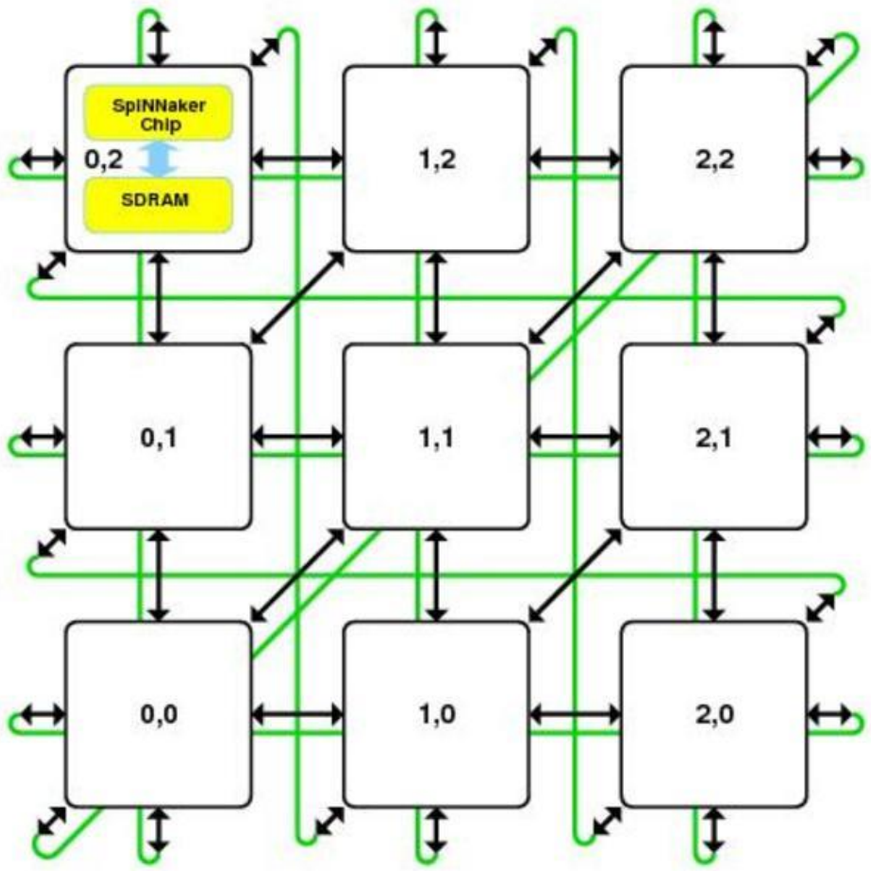
Multiple boards can be connected through SATA to allow further parallel processing exploitation.

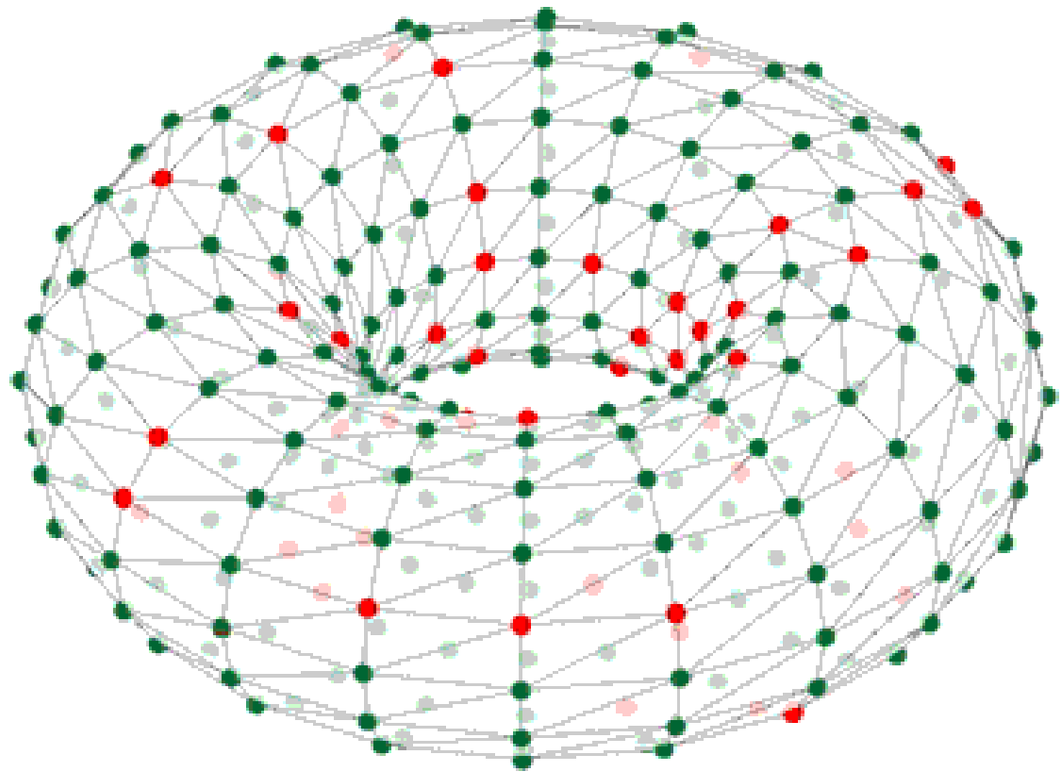
Current largest setup:

- 120 boards per cabinet
- ~500000 cores
- 50 kW peak consumption
- 2.5 million neurons simulated in real-time

Neuromorphic hardware – SpiNNaker

In order to provide fast spike transmission between cores, a proper connectivity method must be exploited.





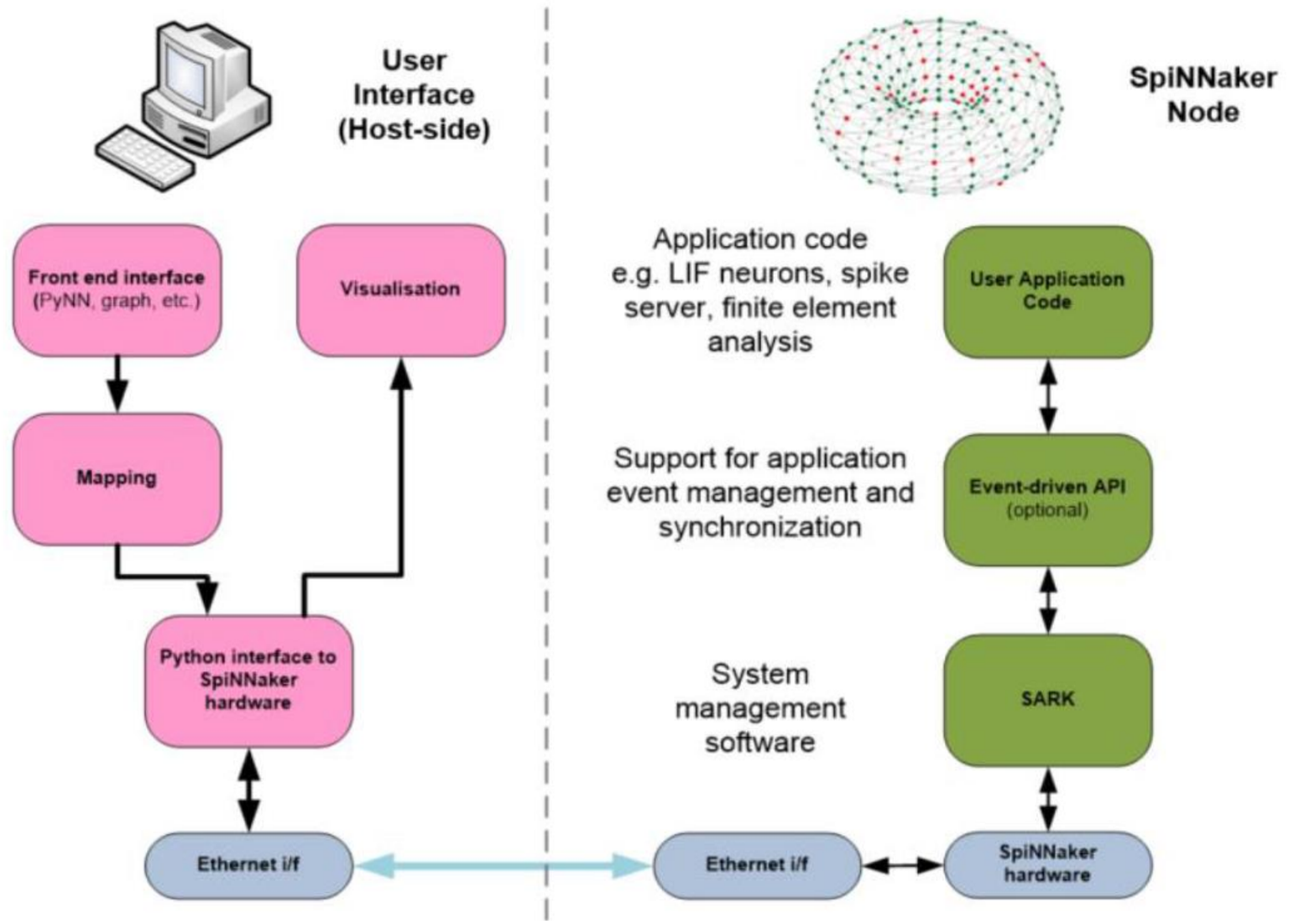
Toroidal connectivity ensures fast spike delivery among chips.

Neuromorphic hardware – SpiNNaker

How do one use these boards? There is a Python library that we can use to set up the network on the SpiNNaker cores: PyNN.

PyNN is a frontend for different neural simulators (including SpiNNaker and NEST)

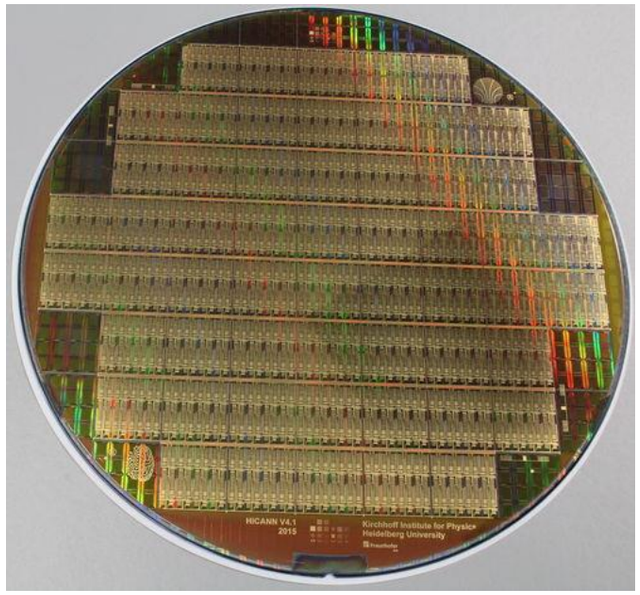
neuralensemble.org/PyNN

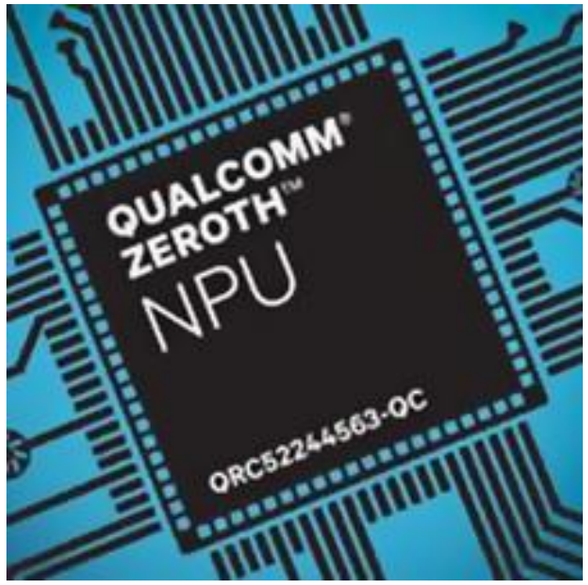


Neuromorphic hardware

SpiNNaker is not the only neuromorphic hardware platform:

Name	Developer	Features
TrueNorth	IBM	Custom processor, 4096 cores with 256 neurons each
BrainScaleS	Heidelberg	Physical model, accelerated simulation time
Brainstorm	Stanford	Physical model, real time
Zeroth	Qualcomm	Deep learning on Snapdragon





Neuromorphic hardware – Final remarks

Pros:

- **real time** neural simulation
- low power consumption
- portable (can be embedded on robots)

Cons:

- cost, availability
- limited number of neurons and connections by design
- still in development
- can lose spikes if firing rates are too high

Suitable to be embedded on a **physical robotic platform**.

Outline

1. Introduction
2. Fundamentals of neuroscience
3. Simulating the brain
4. Software and hardware simulations
5. **Robotic applications**

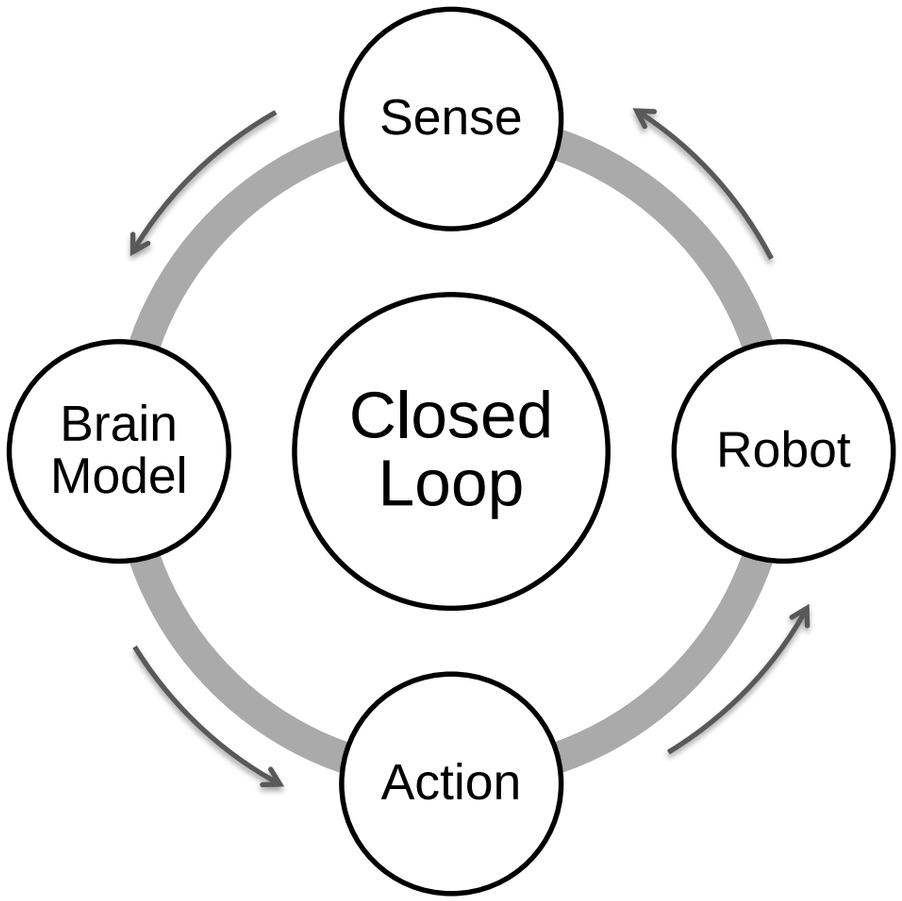
Robotic applications

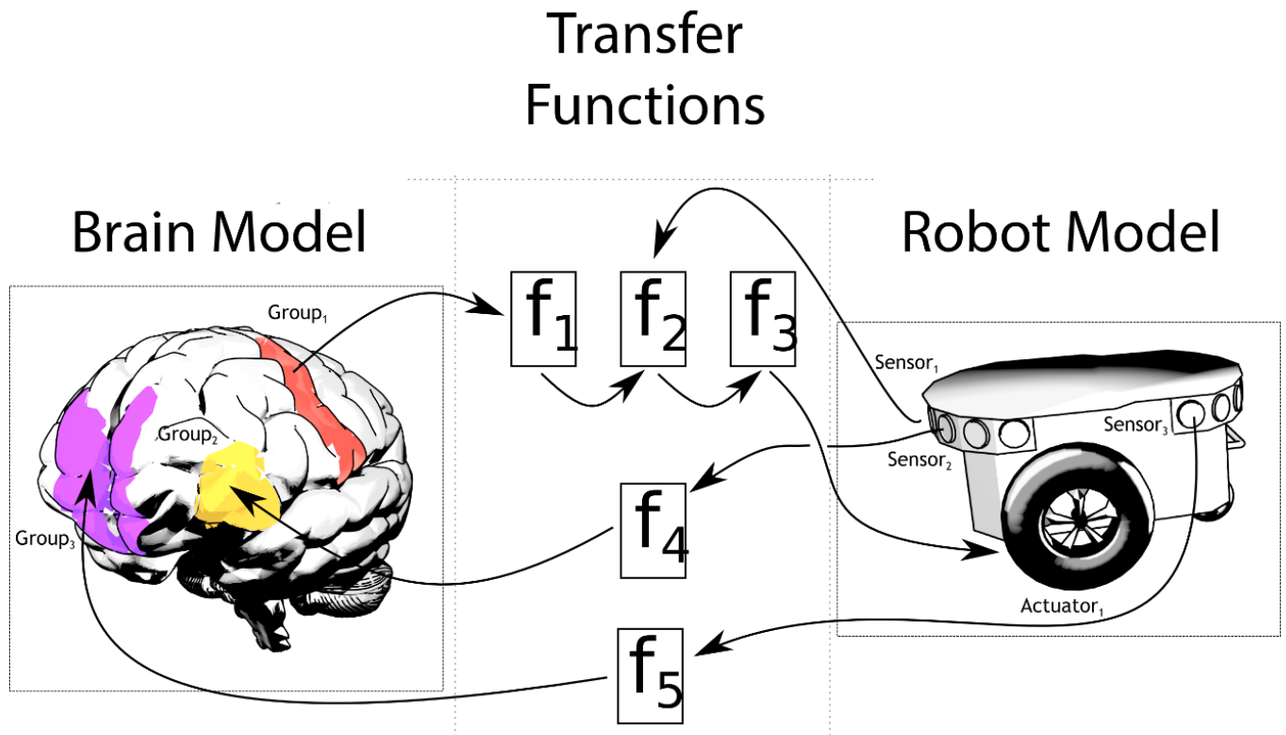
How can we integrate brain models with robotic platforms?

- spiking neural network can be integrated alongside classic robot controllers, relieving them of some computation
- bio-inspired brain models works well for processing of data coming from bio-inspired sensors
- bio-inspired brain models works well for bio-inspired actuators (tendon driven robots, muscle like actuators)
- if connected to a robot, the neural simulation must run in real-time
- if real-time neural simulation is not possible we have to simulate also the robot

The Neurorobotic Closed Loop

One way of integrating neuromorphic computing and robotics is implementing a *closed loop*, a complete action-perception mechanism that involves exchanging information between a robot and a brain model. Information between the two must be properly processed and converted.

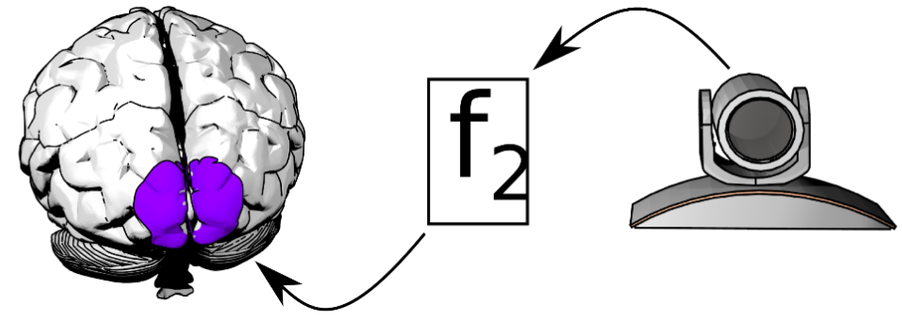




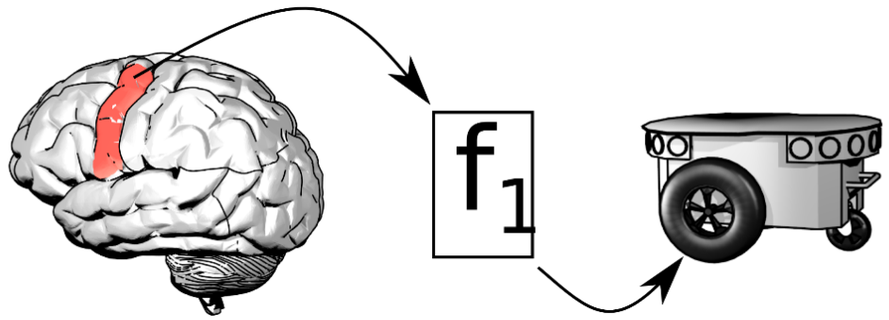
The Neurorobotic Closed Loop

Information between the robot and the brain model must be properly converted and exchanged.

- **robot to neuron**: translate sensory information into spikes and current amplitudes, performing some *enconding*

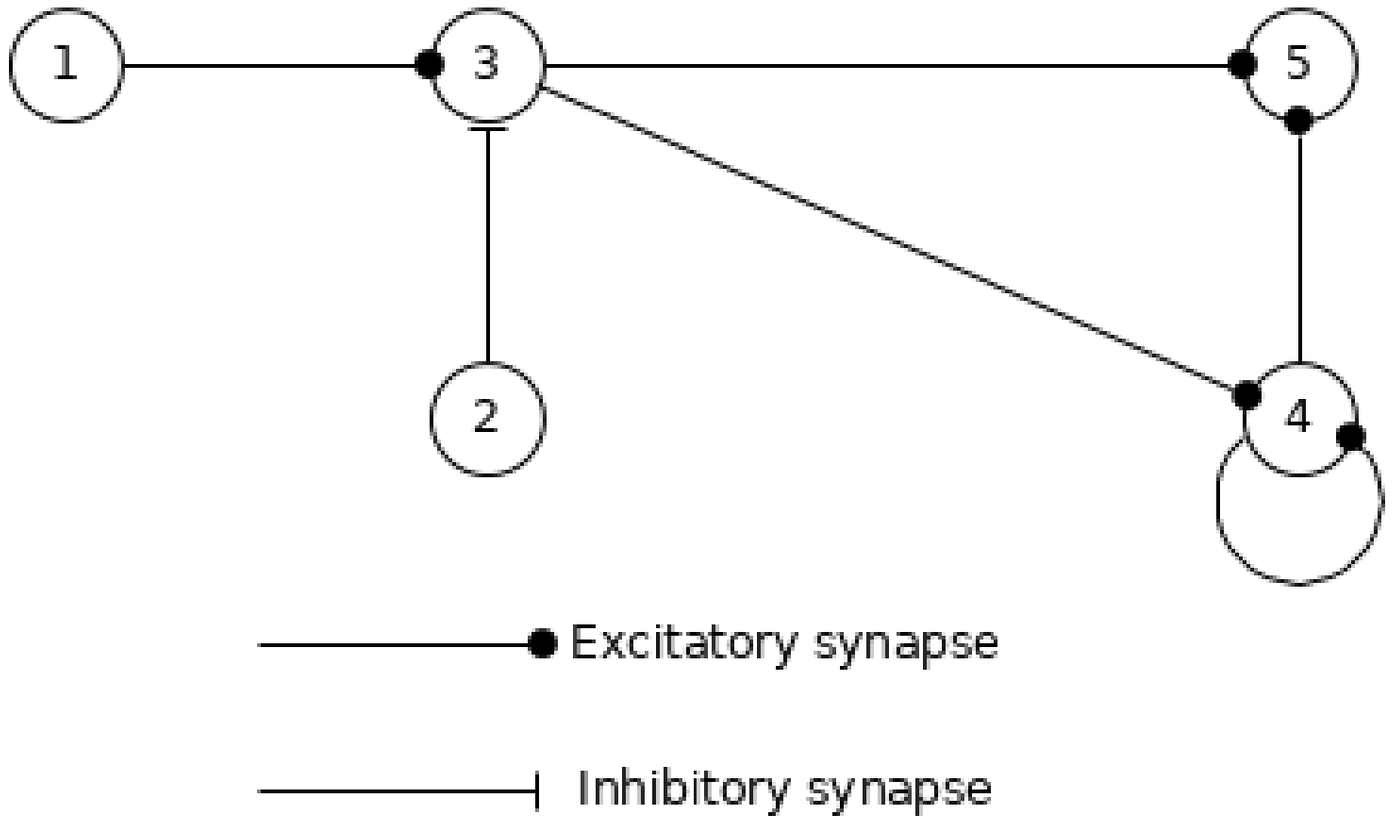


- **neuron to robot**: take measurements on the neural network (spike rate, membrane potential) and transform them into robot commands, thus performing some *decoding*



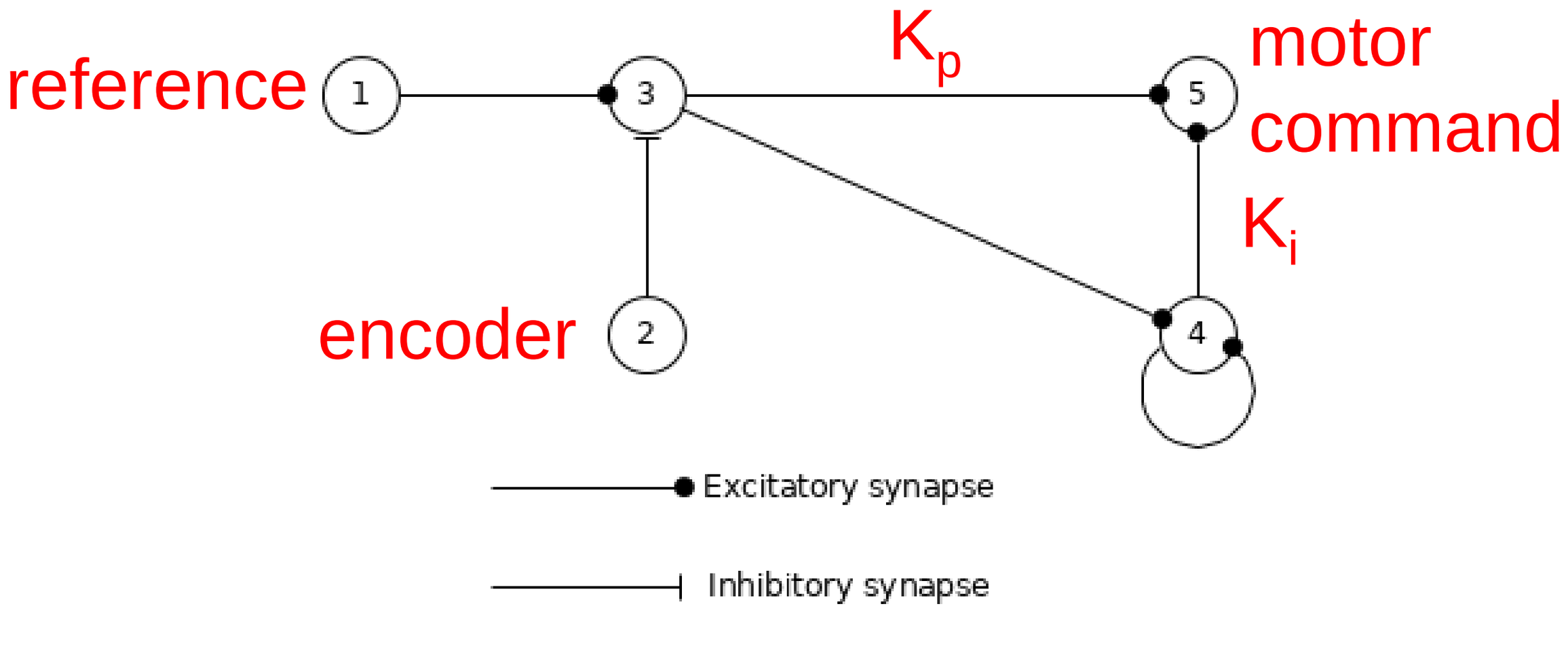
Neurorobotic Closed Loop example

Describe the behaviour of the network below (i.e. what is computed by neuron 5 with respect to neurons 1 and 2). How can such a network be used to implement a low level controller for a motor-actuated robot joint?



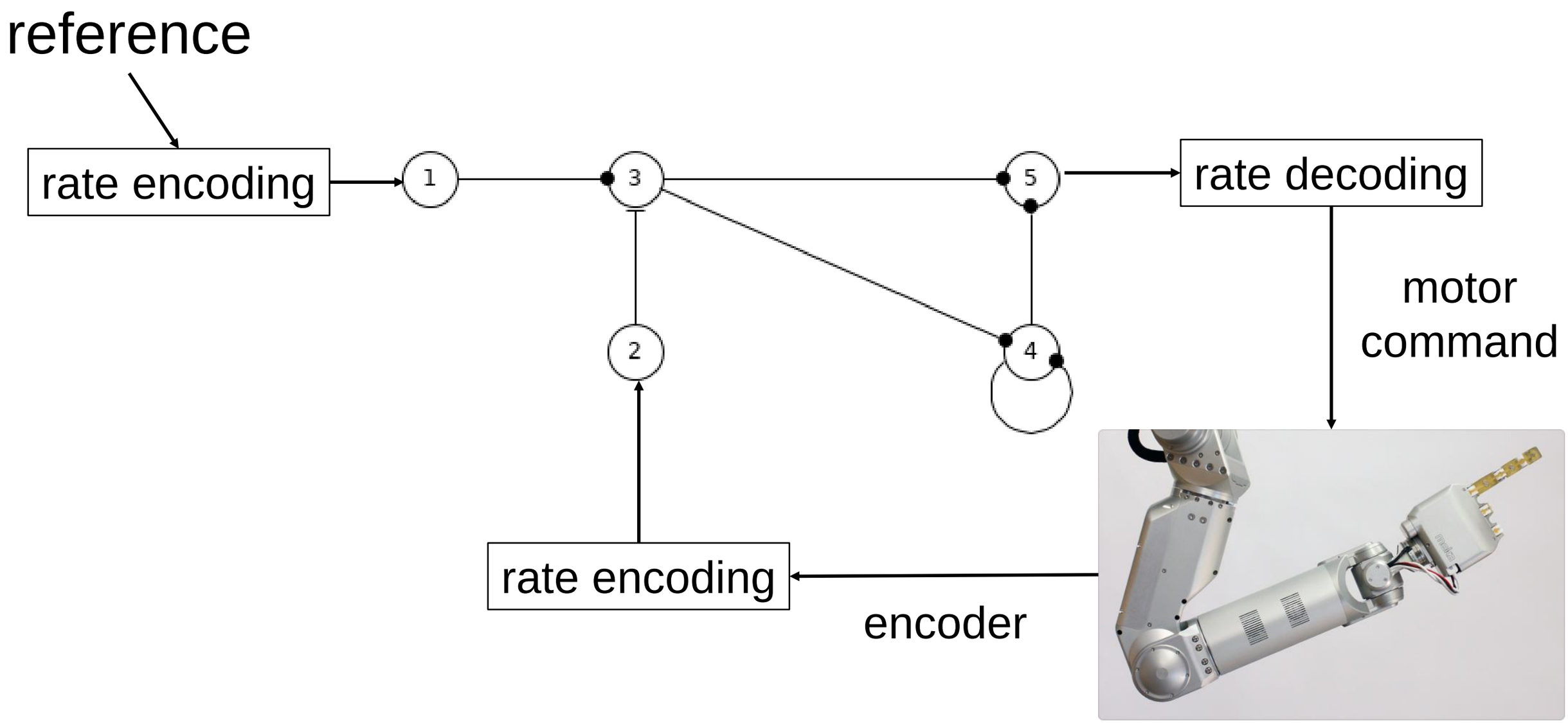
Neurorobotic Closed Loop example

Solution: a PI controller.



Neurorobotic Closed Loop example

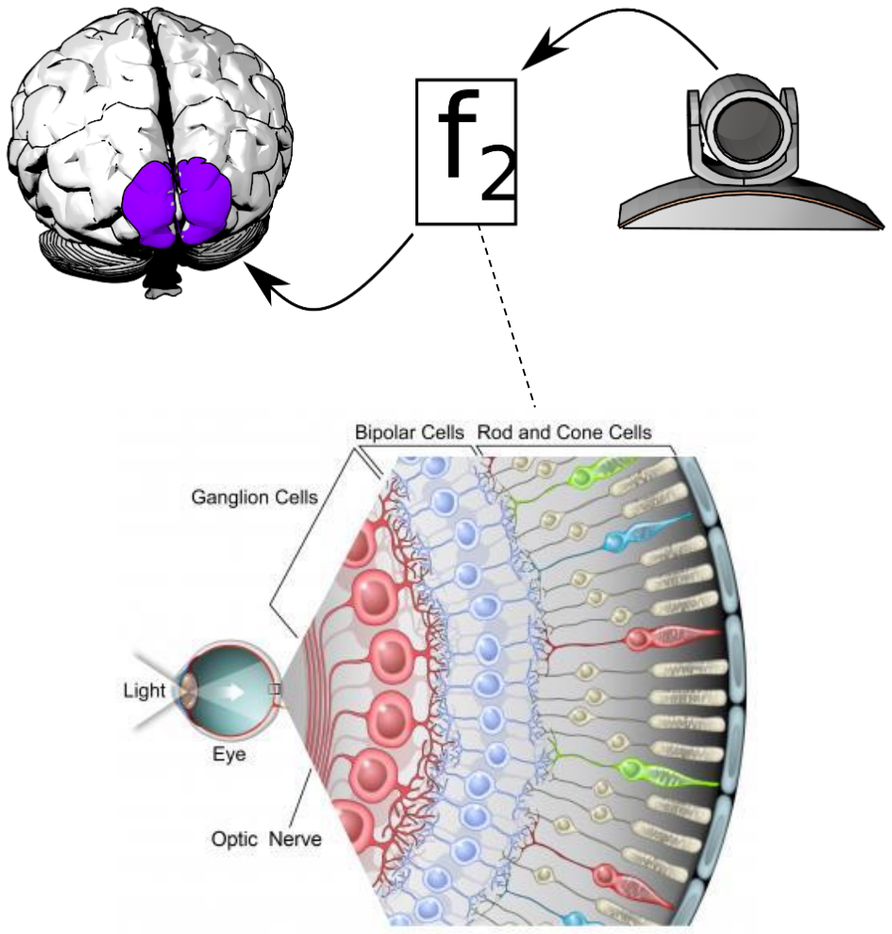
Solution: a PI controller.

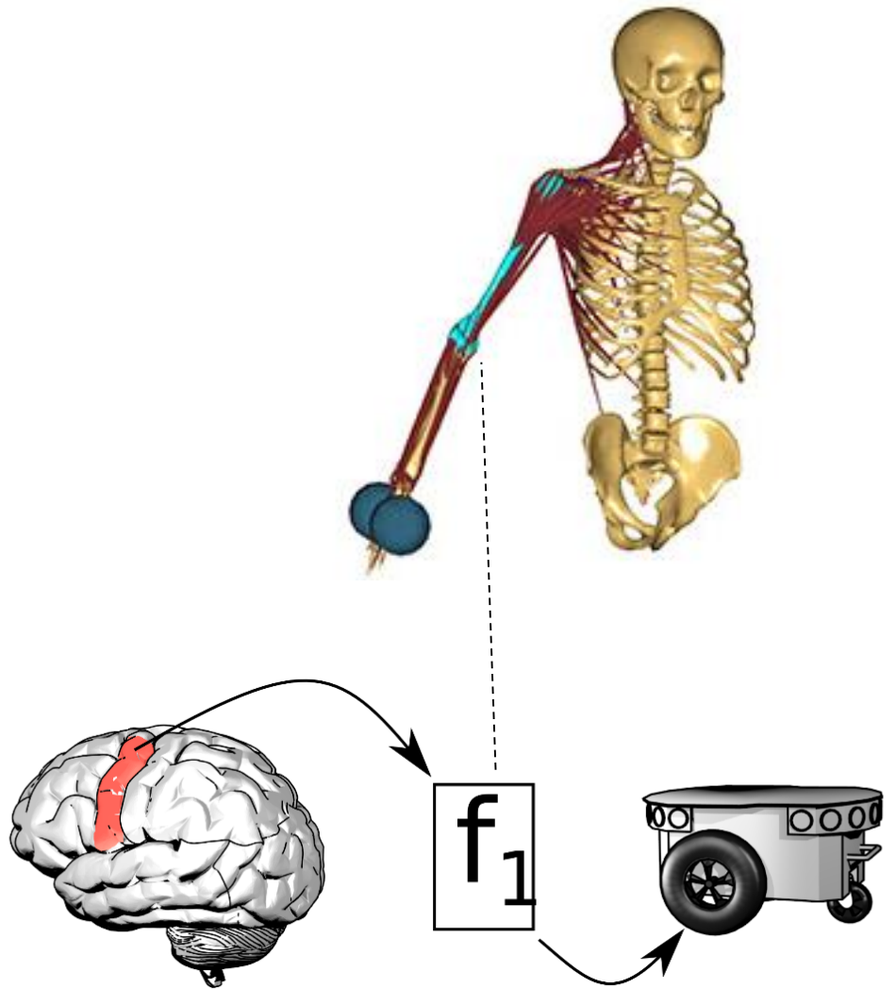


The Neurorobotic Closed Loop

Can we find some general methods to translate information between the two worlds?

Perhaps we could use biological models to do so.

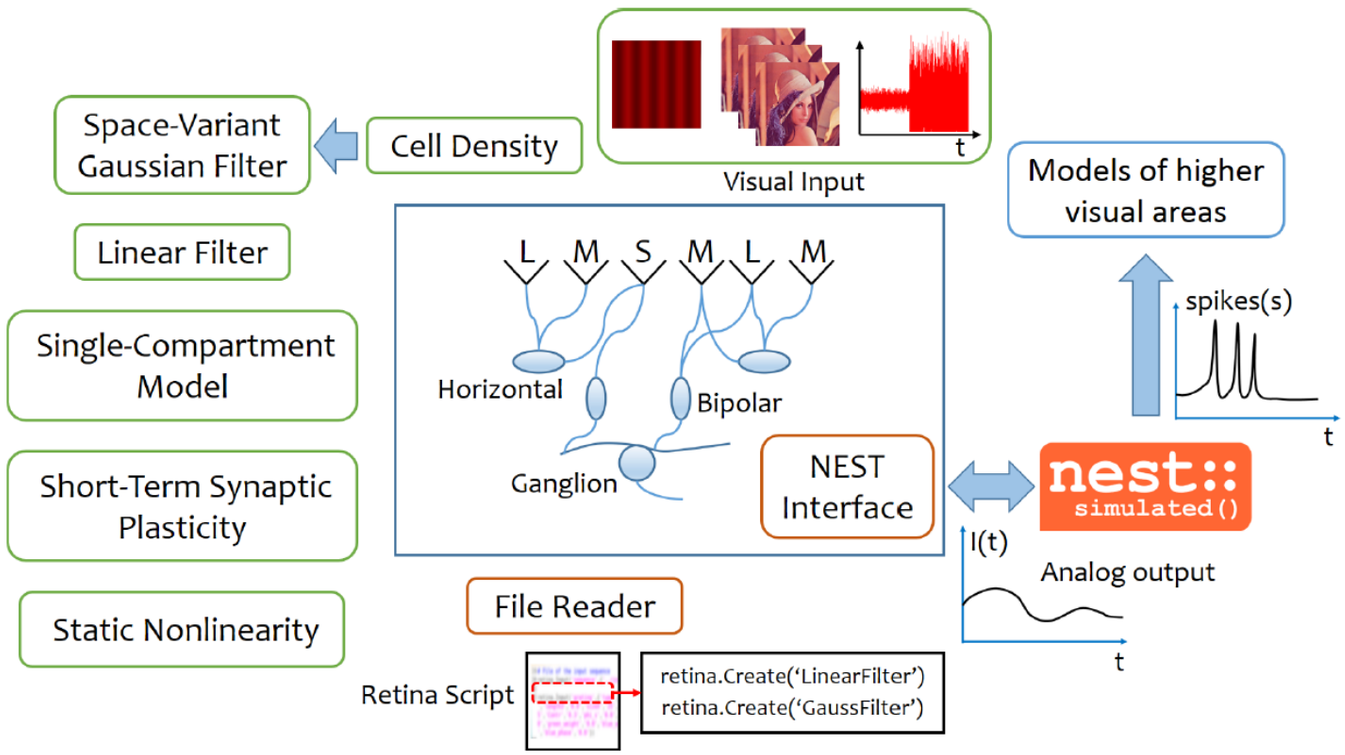




Retinal visual tracking

In this work we integrated bio-inspired sensing with spiking neural network in order to perform a visual tracking task.

We used the same setup as before where we also integrated a retina simulation as a robot to neuron transfer function.



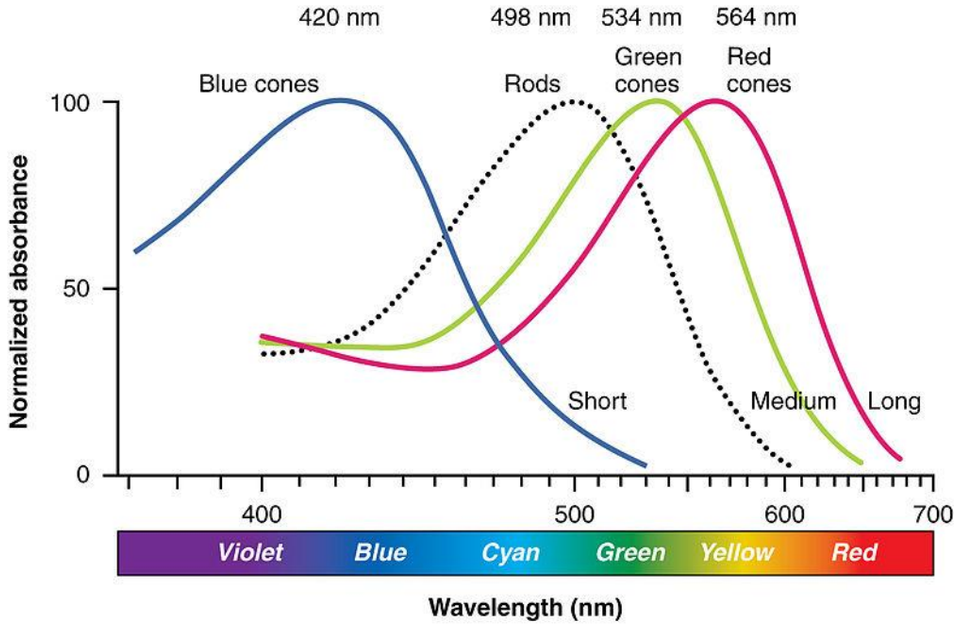
COREM simulator, developed by University of Granada.

- custom retina models
- based on linear/non-linear analysis
- produces an output compatible with NEST, but not spiking

Ambrosano, Alessandro, Lorenzo Vannucci, Ugo Albanese, Murat Kirtay, Egidio Falotico, Georg Hinkel, Jacques Kaiser et al. "Retina color-opponency based pursuit implemented through spiking neural networks in the neurorobotics platform." In *Conference on Biomimetic and Biohybrid Systems*, pp. 16-27. Springer International Publishing, 2016.

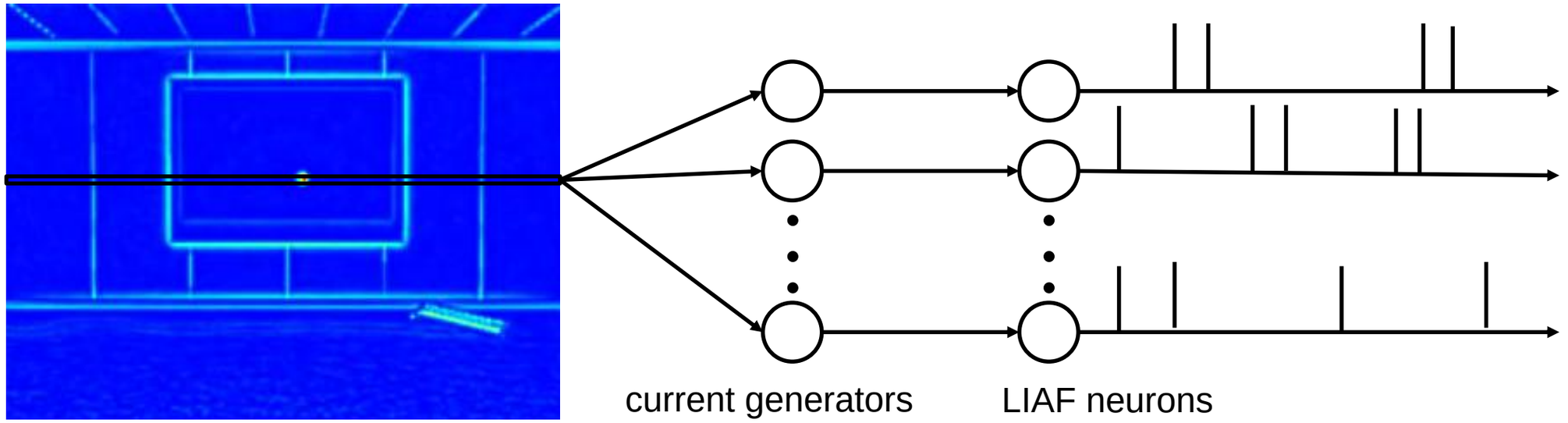
Retinal visual tracking

At first we performed a target detection via retinal image processing.



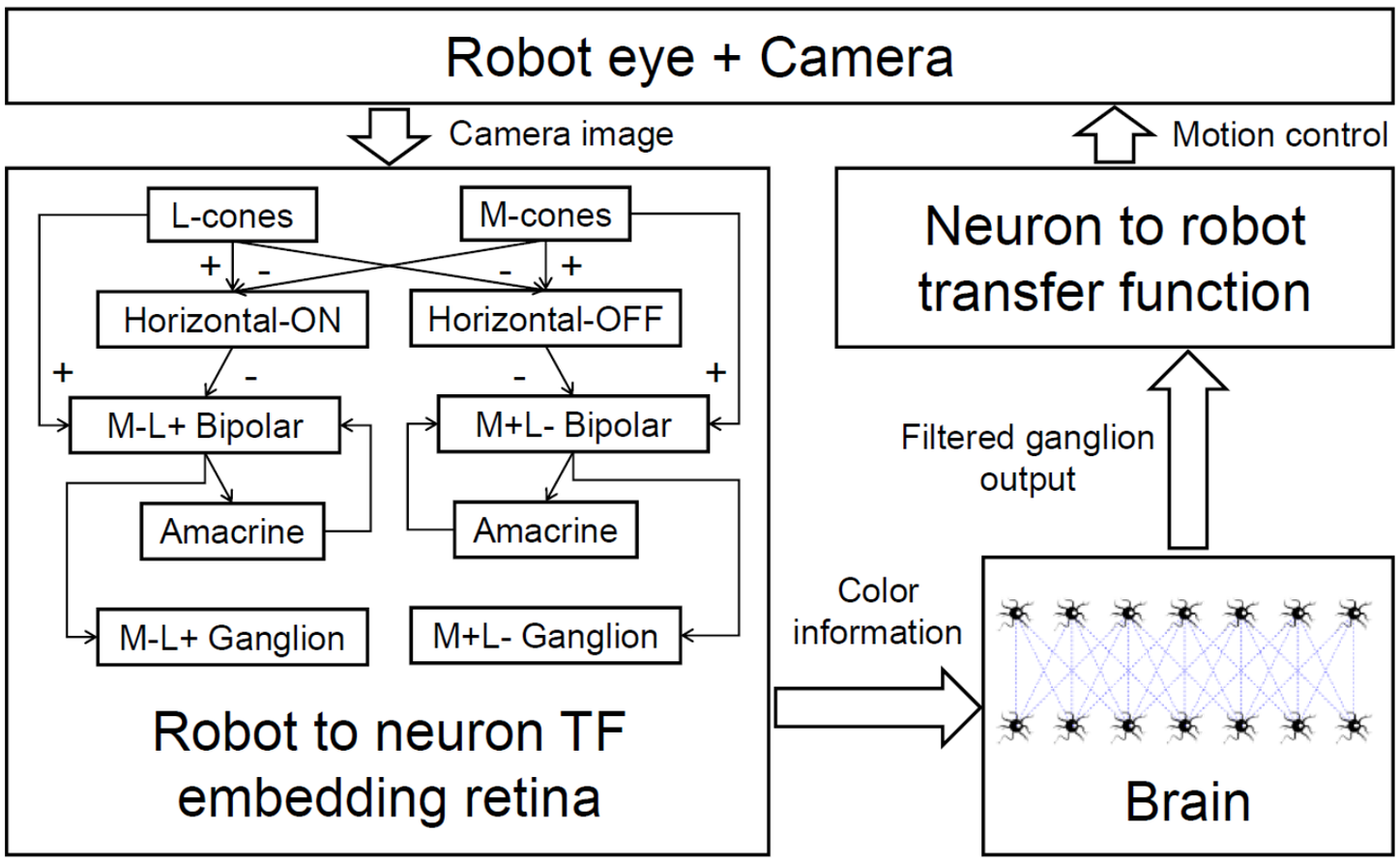
We used a complete retina model, but only with the pathway coming from M-cones (more sensitive to green).

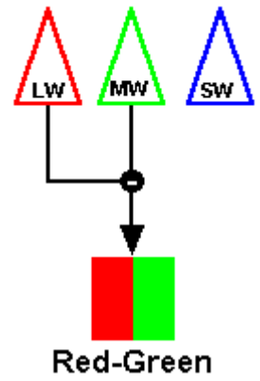
Analogue output from the ganglion cells is sent to a LIAF neuron layer via current generators devices.



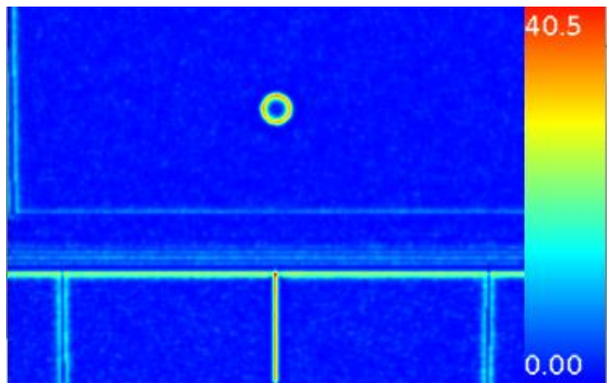
Retinal visual tracking

Then, we switched to a more sophisticated retina model, based on red-green opponency.



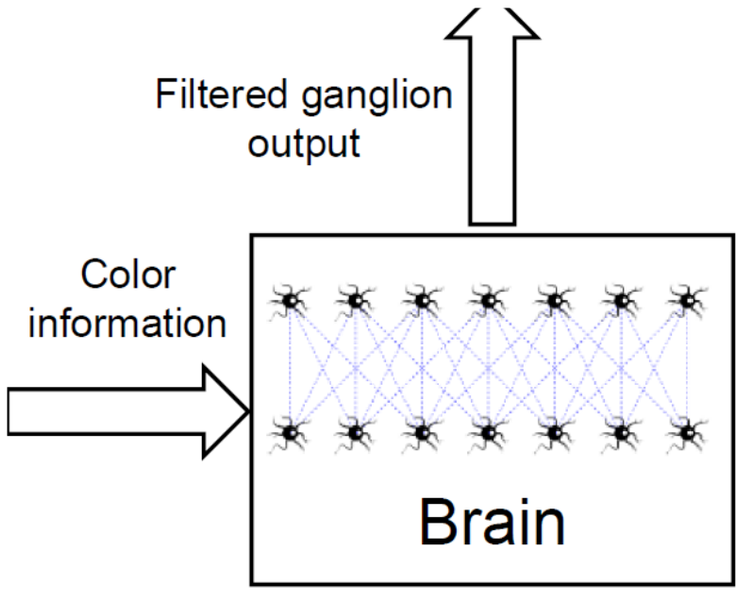


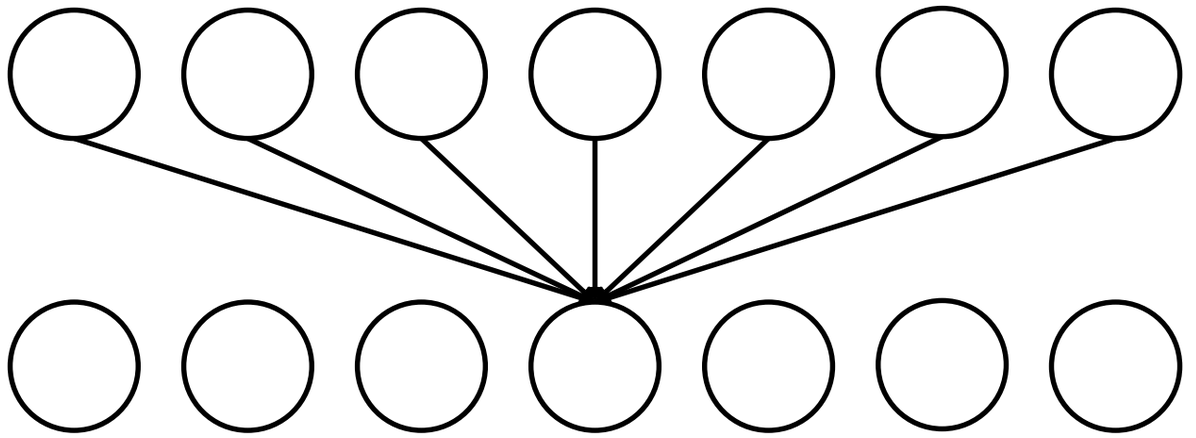
The output value of this retina model is higher for edges of the target.



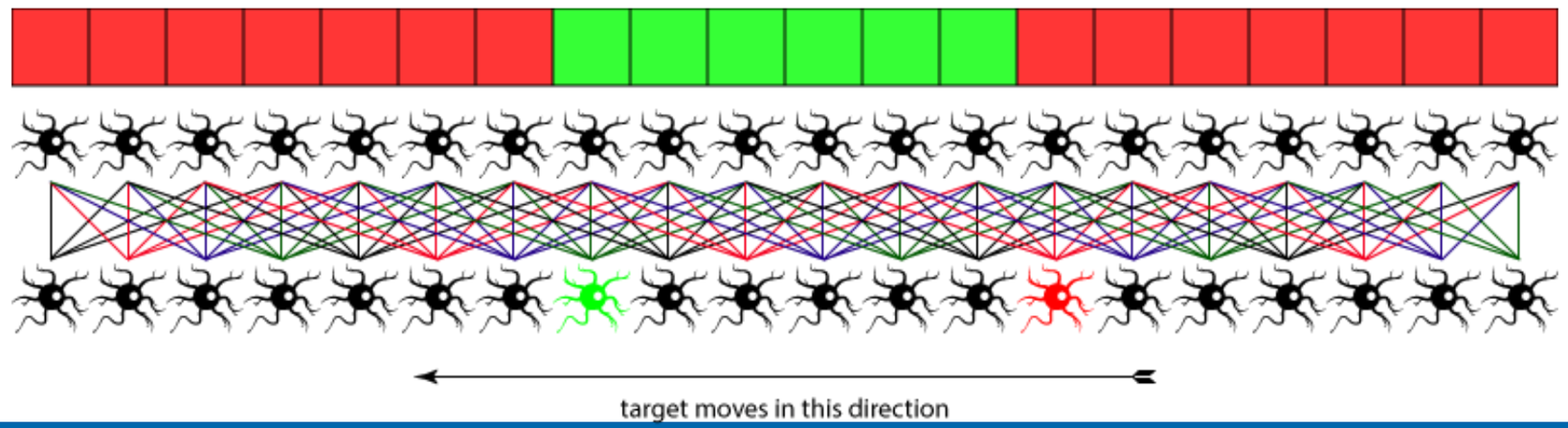
Retinal visual tracking

In order to filter out the noise from the ganglion output and retain only the target information, a two layer spiking neural network was used.



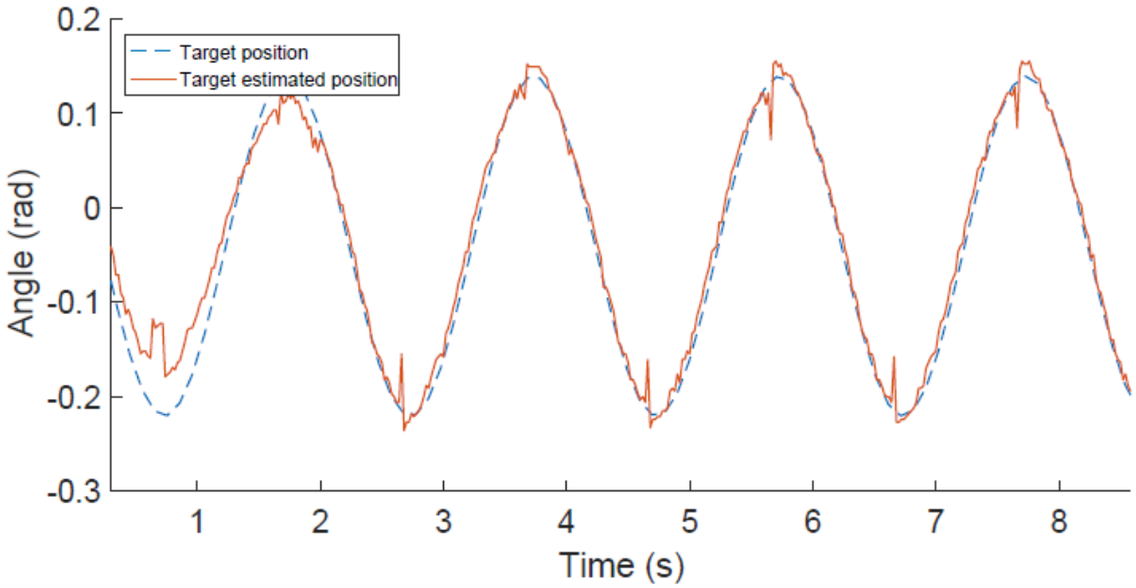


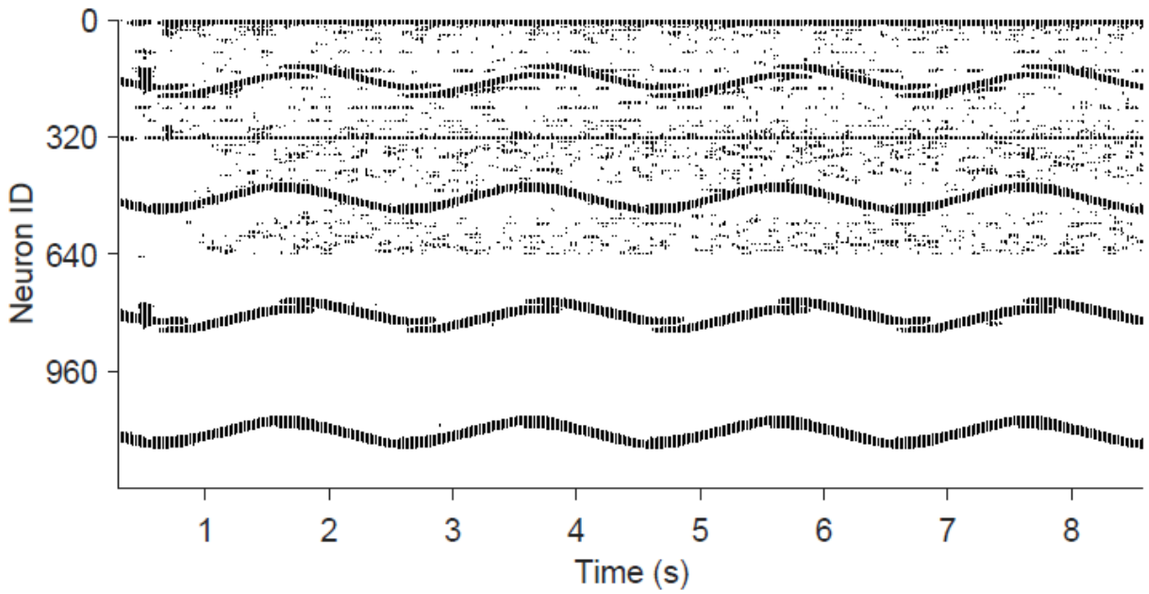
The first layer is a current to spike converter. Neurons in the second layer receives spikes from a *receptive field* of 7 neurons (pixels).



Retinal visual tracking

Using output of the neural network we can estimate the target centroid and use this information to generate motor commands for the robot eye.

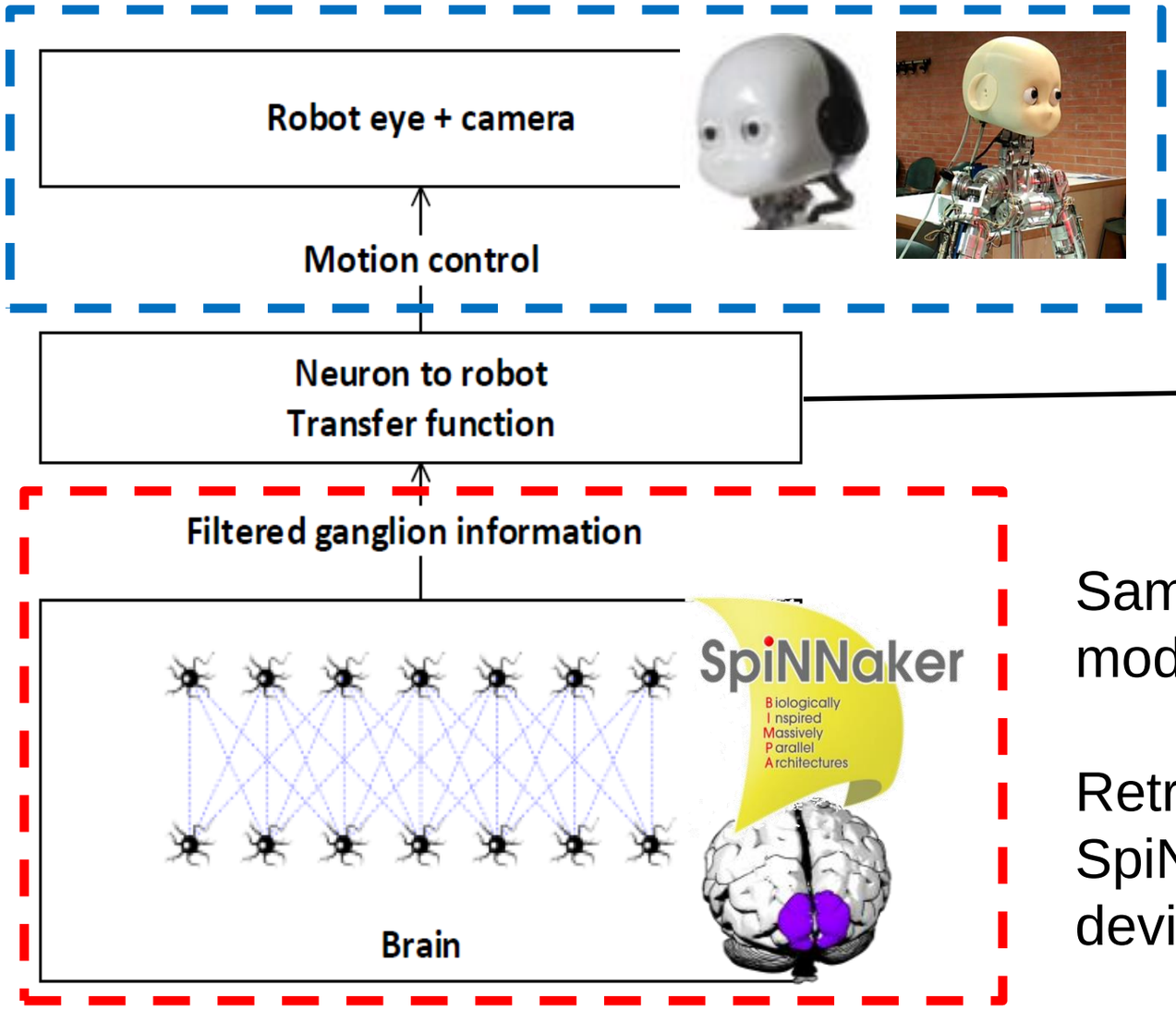




The robot is able to follow the target thanks to the neural filtering of the retinal output.

Visual tracking with SpiNNaker

The same controller was also implemented on the real iCub robotic platform, using neuromorphic hardware for real-time neural simulation.



Real iCub robot, accessed via YARP

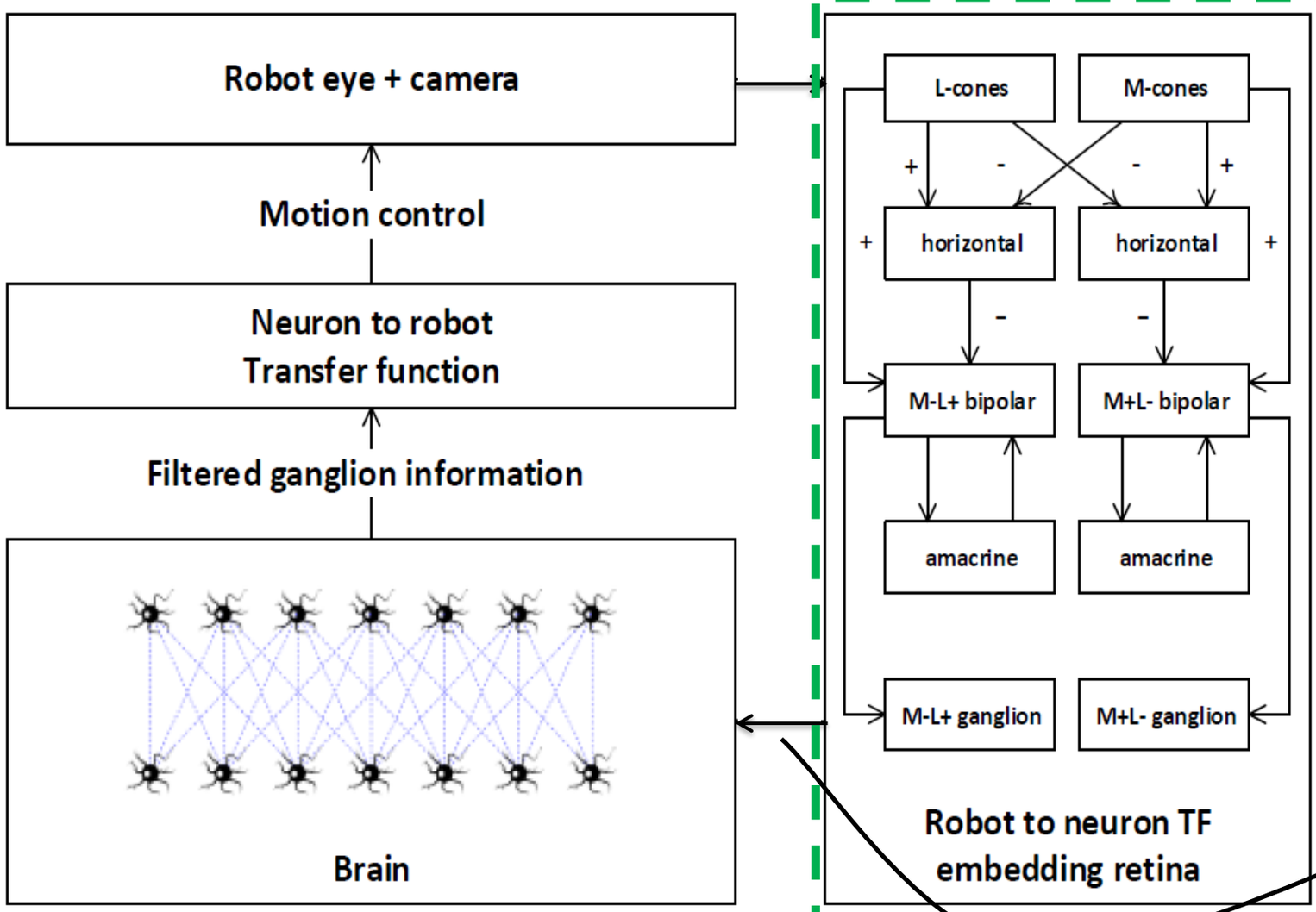
Same transfer function

Same PyNN brain model

Retrieval of spikes from SpiNNaker receiver devices

Visual tracking with SpiNNaker

The same controller was also implemented on the real iCub robotic platform, using neuromorphic hardware for real-time neural simulation.

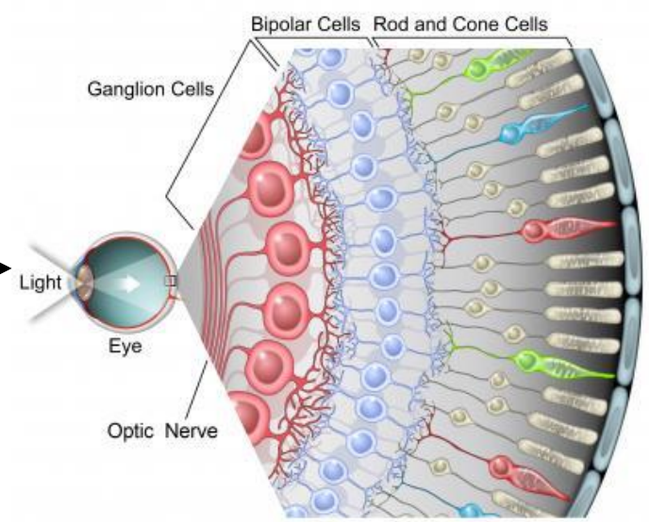


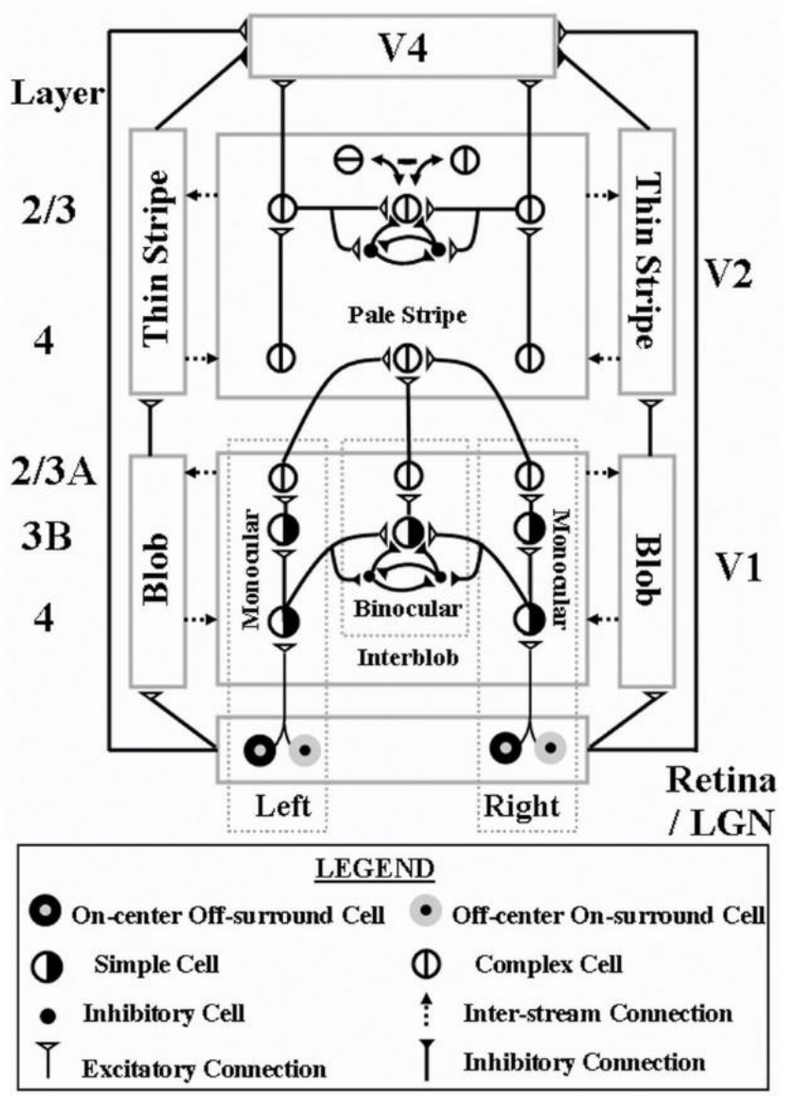
COREM framework for simulating retinal computation.
Processing 320x240 images up to 20Hz.

Simulation of DC generator + IAF neuron dynamics for the generation of spikes times that are sent through a SpiNNaker SpikeInjector.

Retina as a generic translation mechanism

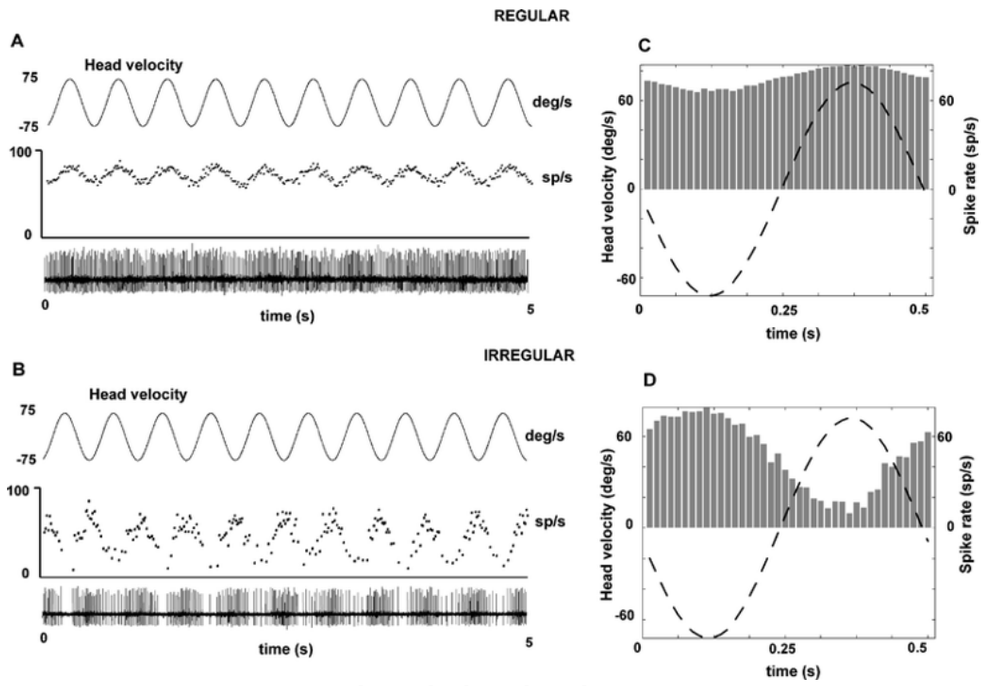
This kind of translation is actually generic and in fact it was employed with a more complex visual cortex model.



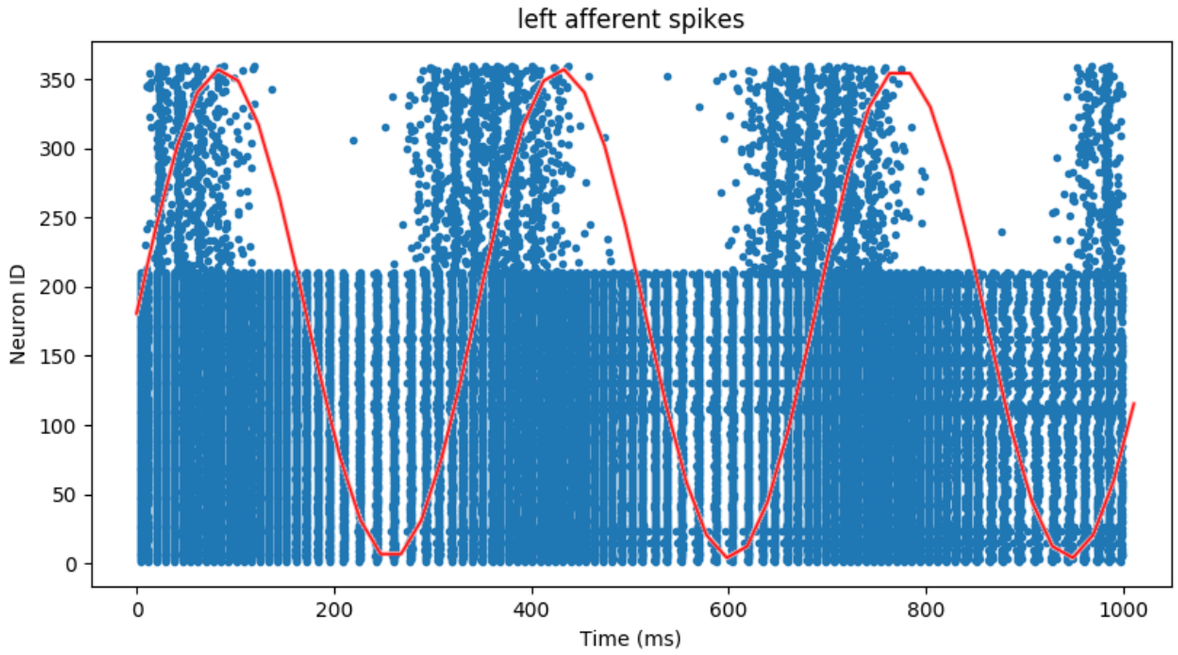


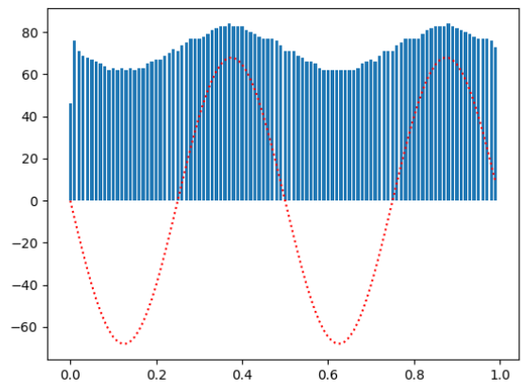
Neuromorphic model of vestibular afferents

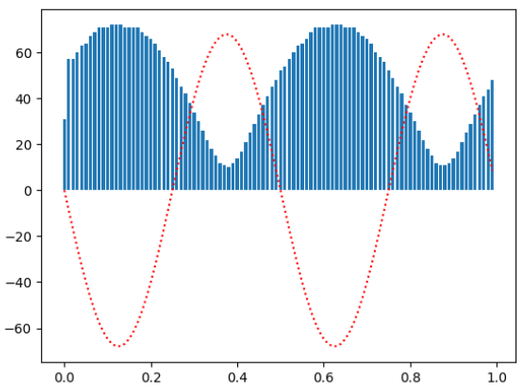
In order to translate information coming from inertial sensors, we developed a neuromorphic model of vestibular afferents that comprises of both **regular** and **irregular** afferents.



Neurophysiological recordings







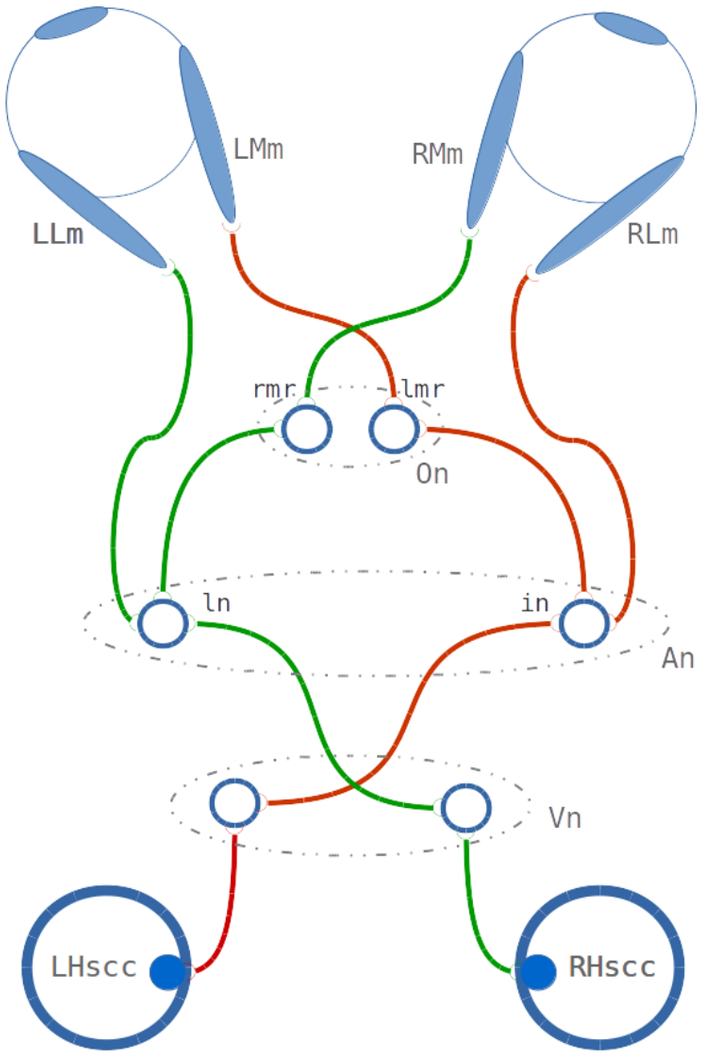
NEST implementation

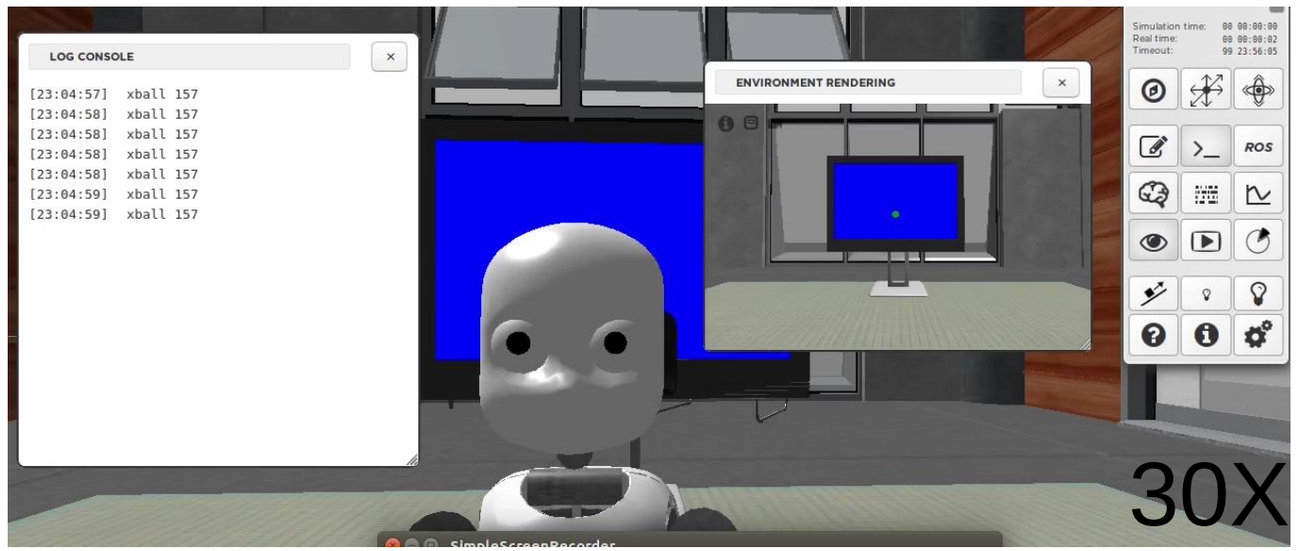
$$I(t) = G_H \cdot HV(t) - G_A \cdot X_A(t) + I_{bias} + \sigma\epsilon(t)$$

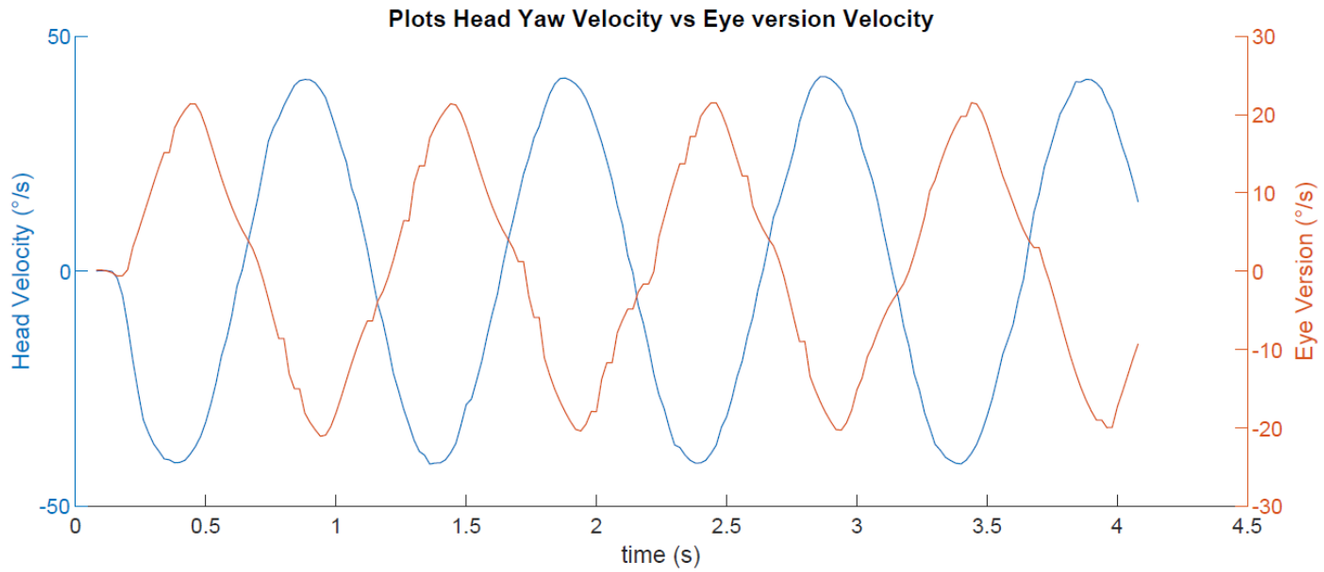
$$\frac{dV}{dt} = \frac{V(t) + I(t)}{\tau_V}$$

Neuromorphic model of vestibular afferents

To test the effectiveness of the model, a complete spiking network implementing the VOR circuit was for the iCub robot.

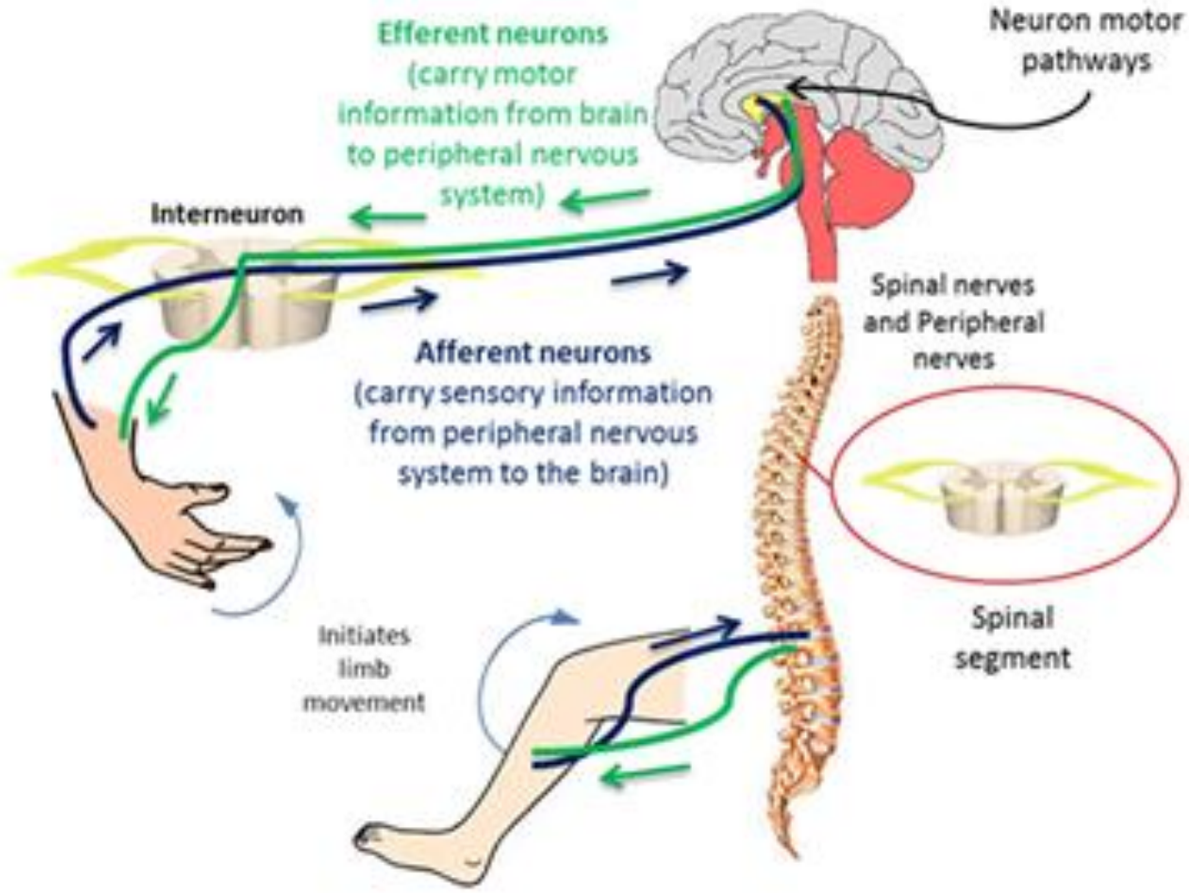






Robot-brain connection through a spinal cord model

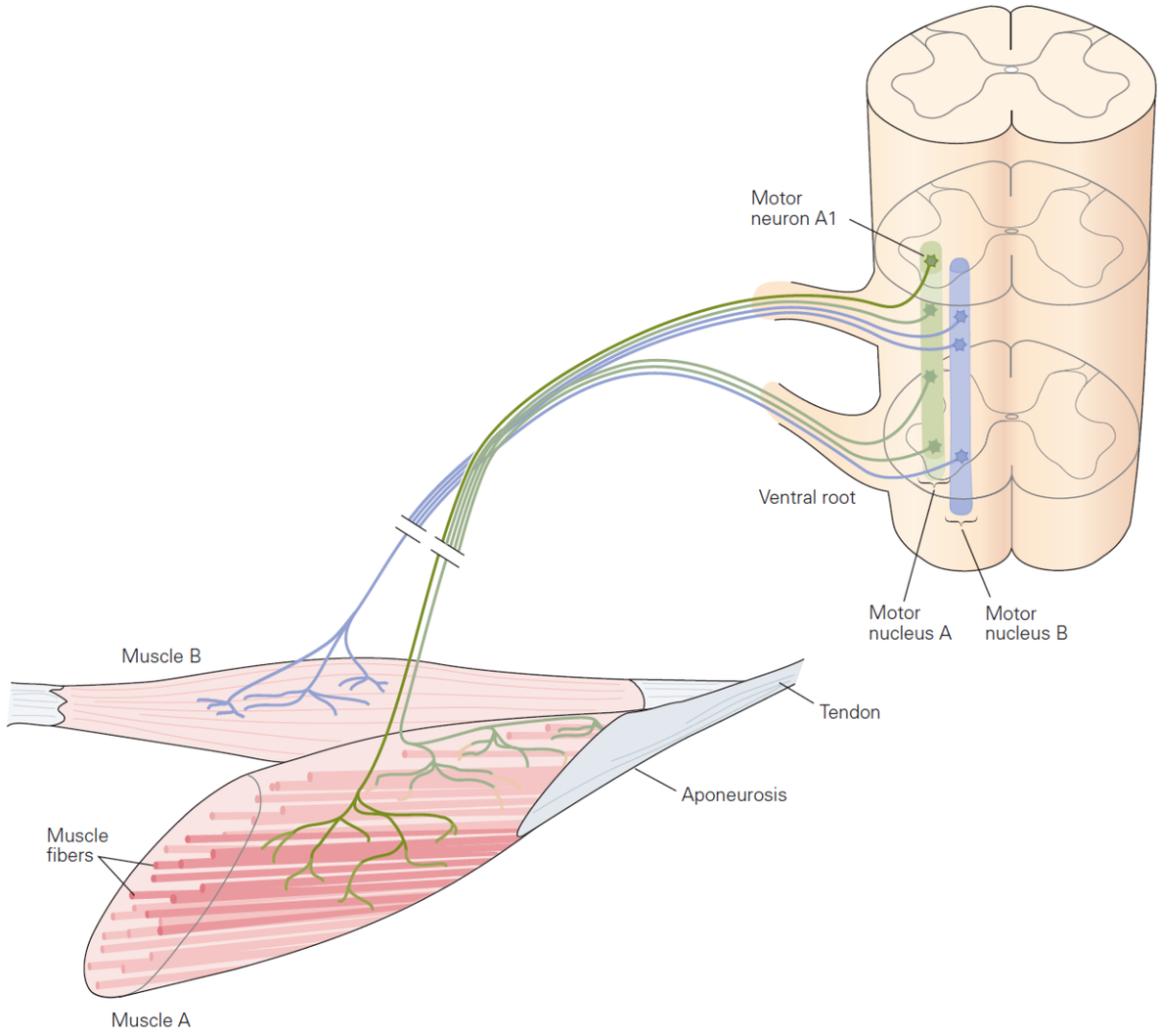
In most animals, motor commands from the brain cortex are not directly sent to the muscles, but they are transmitted through a series of hierarchically organized neural circuits. At the lowest level of this hierarchy lies the spinal cord.

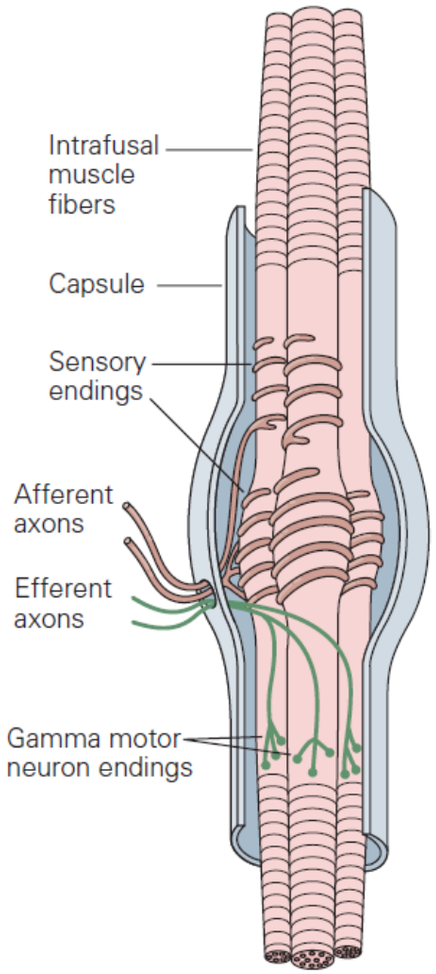


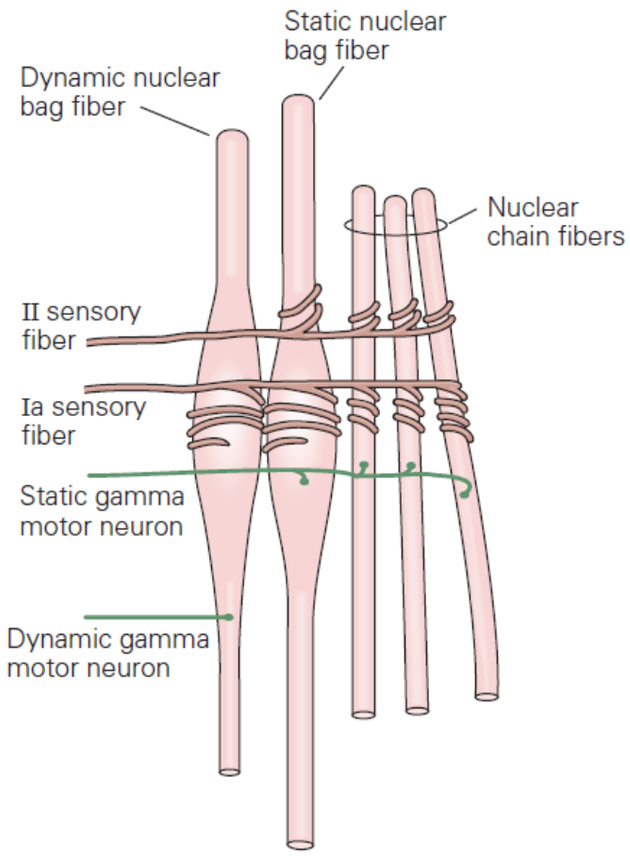
Therefore, we can think of implementing a spinal cord model that performs the translation of proprioceptive feedback and the generation of motor commands.

Robot-brain connection through a spinal cord model

The spinal cord contains *α-motoneurons* that directly activate the muscle fibres, as well as sensory feedback endings such as *Ia* and *II* afferents from *muscle spindles*.

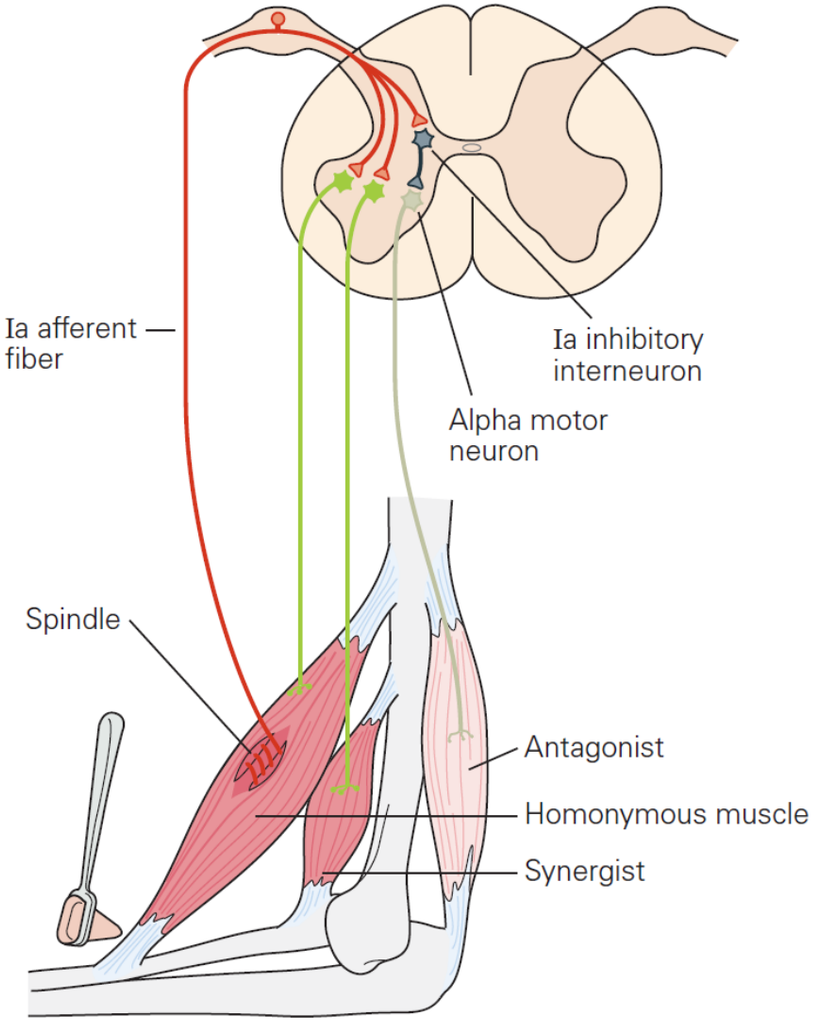


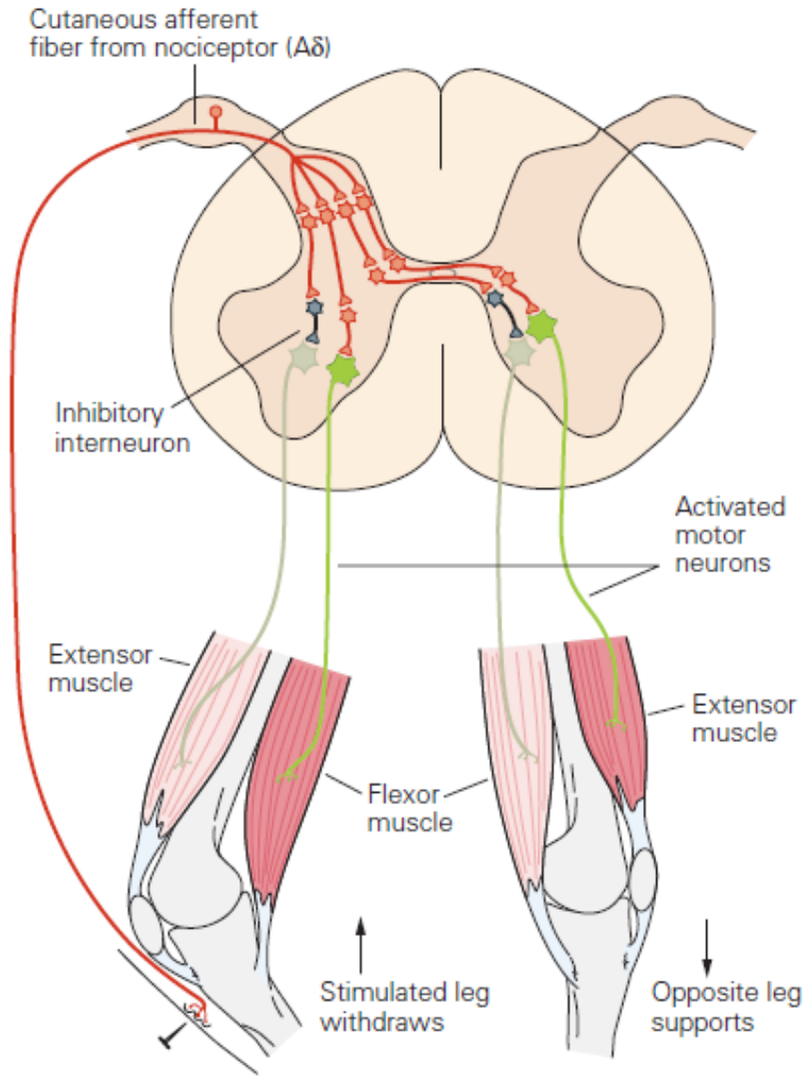




Robot-brain connection through a spinal cord model

The spinal cord is not only responsible for the activation of muscles and for the forwarding of proprioceptive information, but it also includes many local circuits for the generation of reflexes.



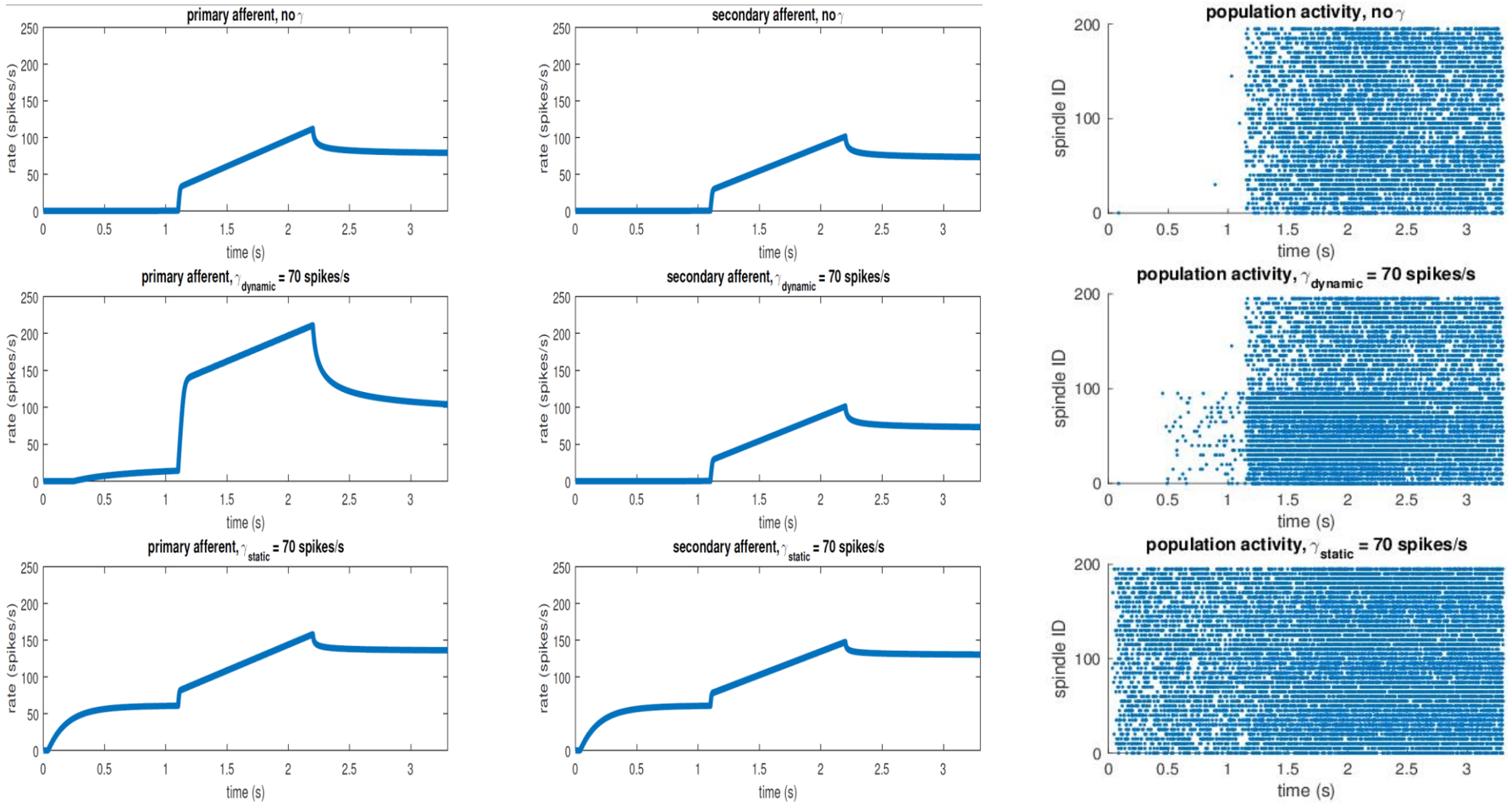


Robot-brain connection through a spinal cord model

We started by implementing in NEST (and on SpiNNaker) a bioinspired model of muscle spindle that simulates Ia and II afferent activities during a muscle stretch.

$$rate \propto T$$

$$\frac{dT}{dt} = \mathrm{f}\left(\mathrm{L}, \frac{dL}{dt}, \gamma_{st}, \gamma_{dyn}\right)$$



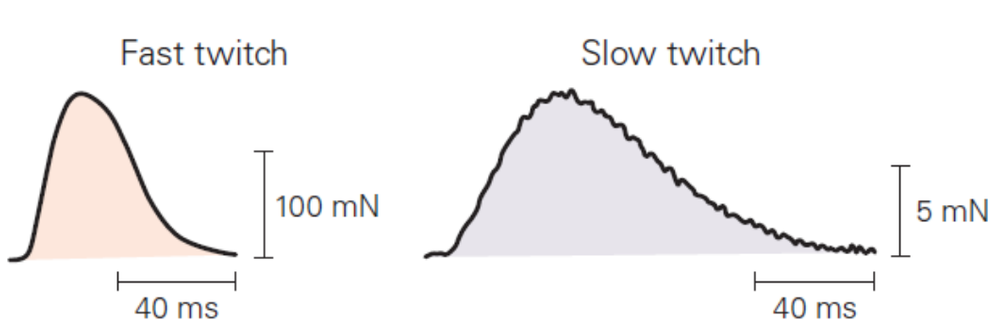
Vannucci, Lorenzo, Egidio Falotico, and Cecilia Laschi. "Proprioceptive Feedback through a Neuromorphic Muscle Spindle Model." Frontiers in Neuroscience 11 (2017): 341.

Robot-brain connection through a spinal cord model

We then implemented in NEST a muscle activation model that includes motoneurons recruitment and twitches integration.

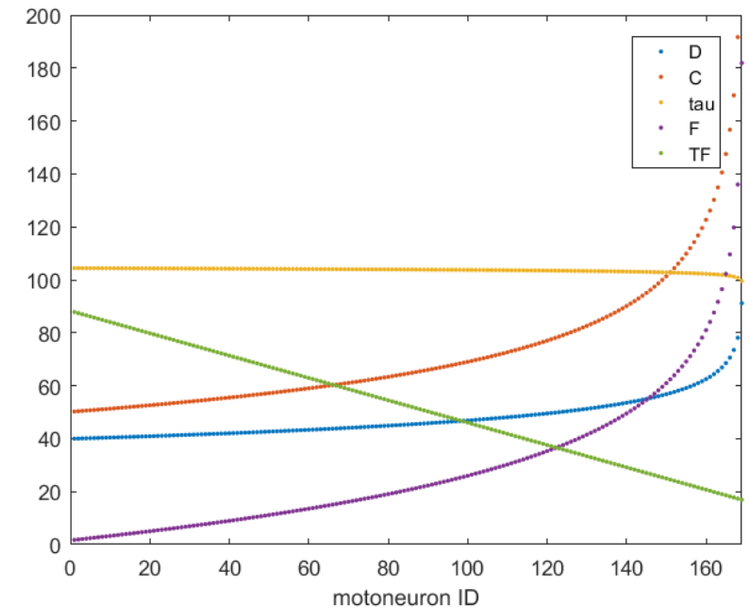
- motoneurons are activated from the weakest to the strongest

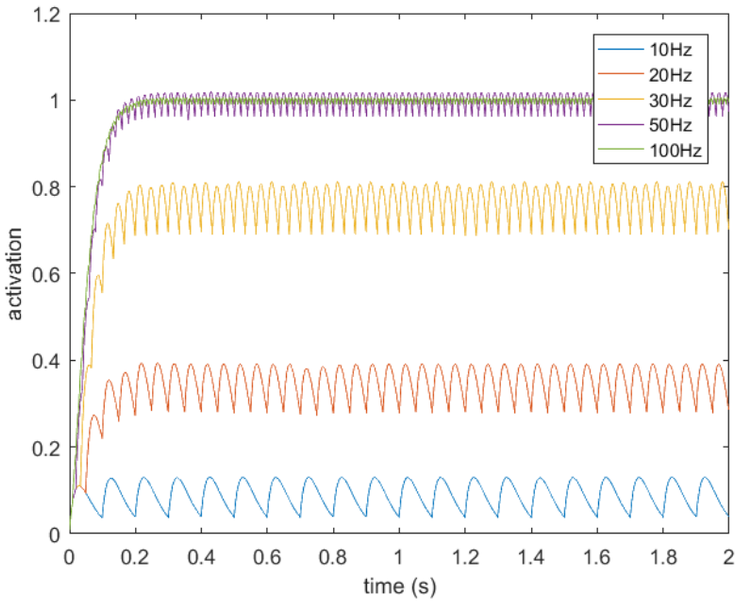
- each activation produces a twitch of some fibres



- all the twitches are summed up to compute the total muscle activation

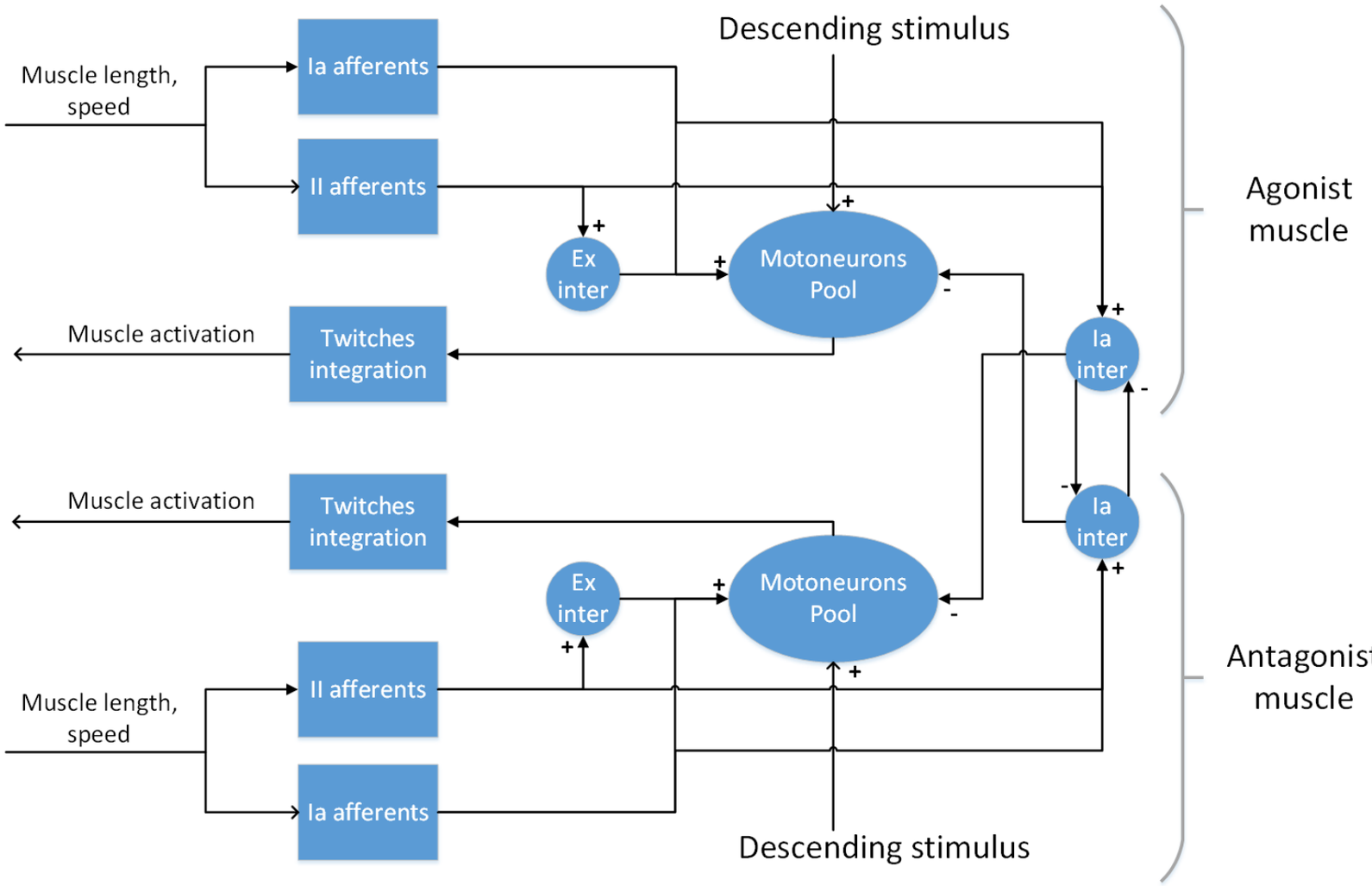
- output can be normalized between 0 and 1





Robot-brain connection through a spinal cord model

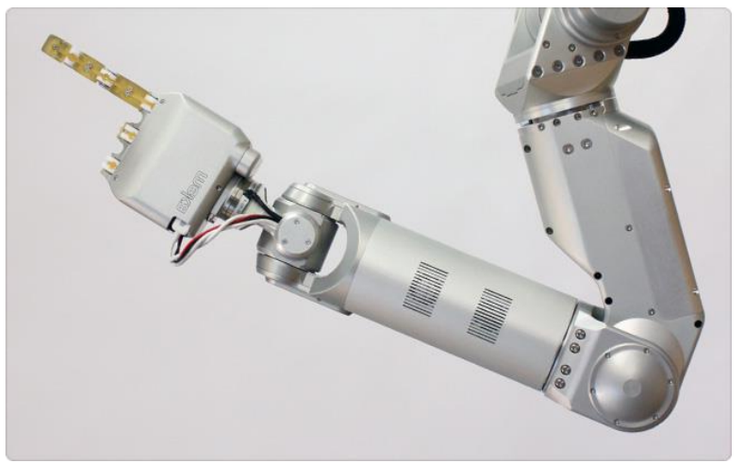
Once we have the basic components we can assemble a fairly complete spinal cord model.



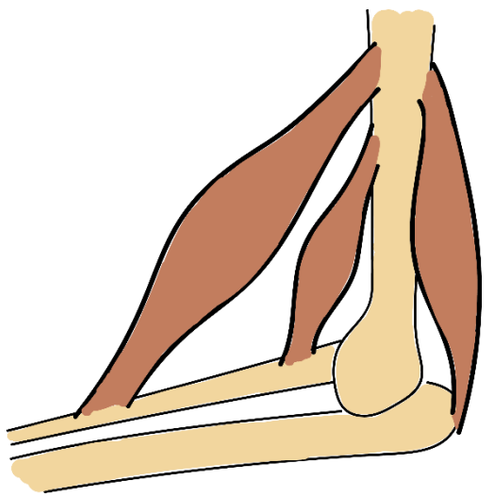
But we don’t know yet how to connect it to a robot…

Robot-brain connection through a spinal cord model

In order to connect it to a robot, the more natural way is to add musculoskeletal system to the robot. Let’s consider a single joint of the robot (elbow joint).



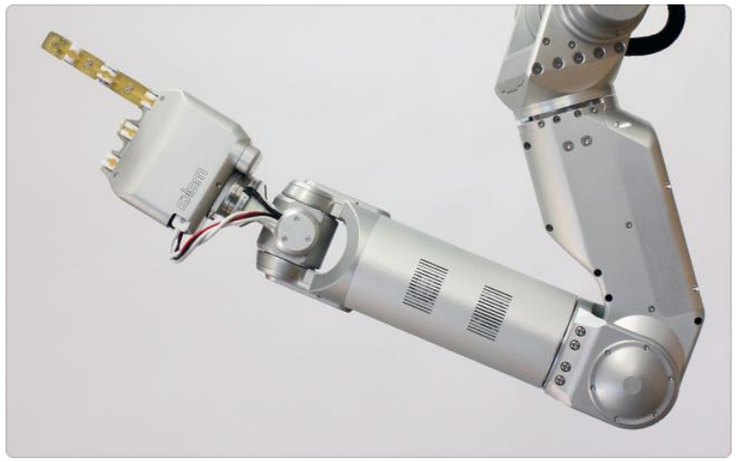
+

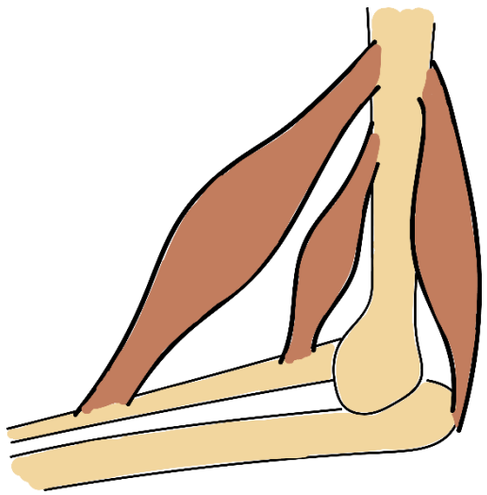


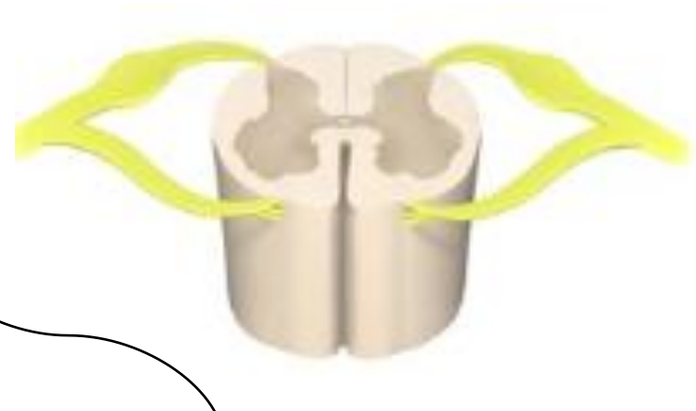
+

Robot-brain connection through a spinal cord model

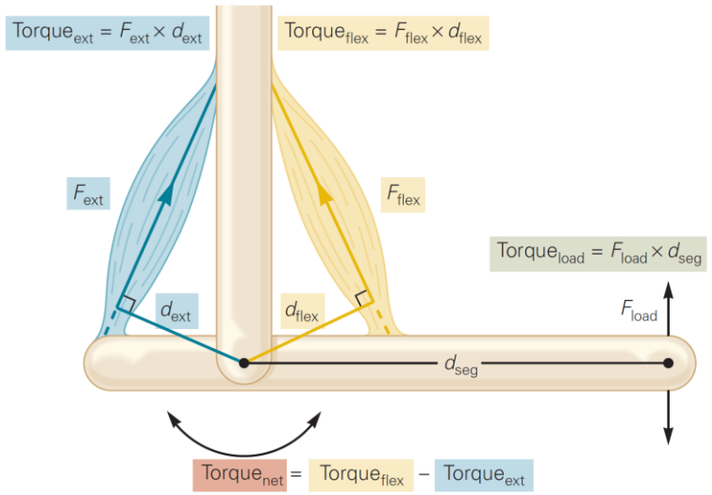
In order to connect it to a robot, the more natural way is to add musculoskeletal system to the robot. Let’s consider a single joint of the robot (elbow joint).



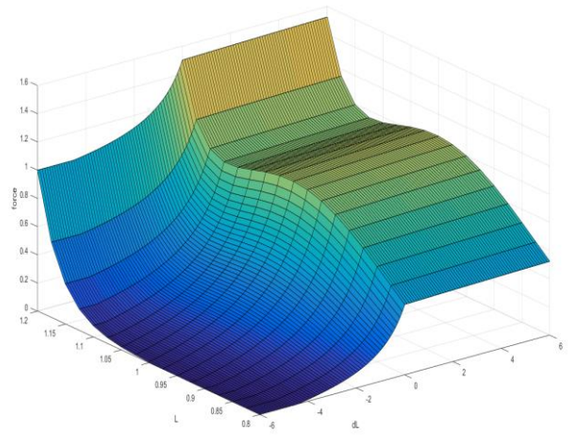




- joint torque from muscle forces
- muscle lengths from joint angle



- muscle forces computed from activations via a Hill model
- muscle lengths sent to spindles

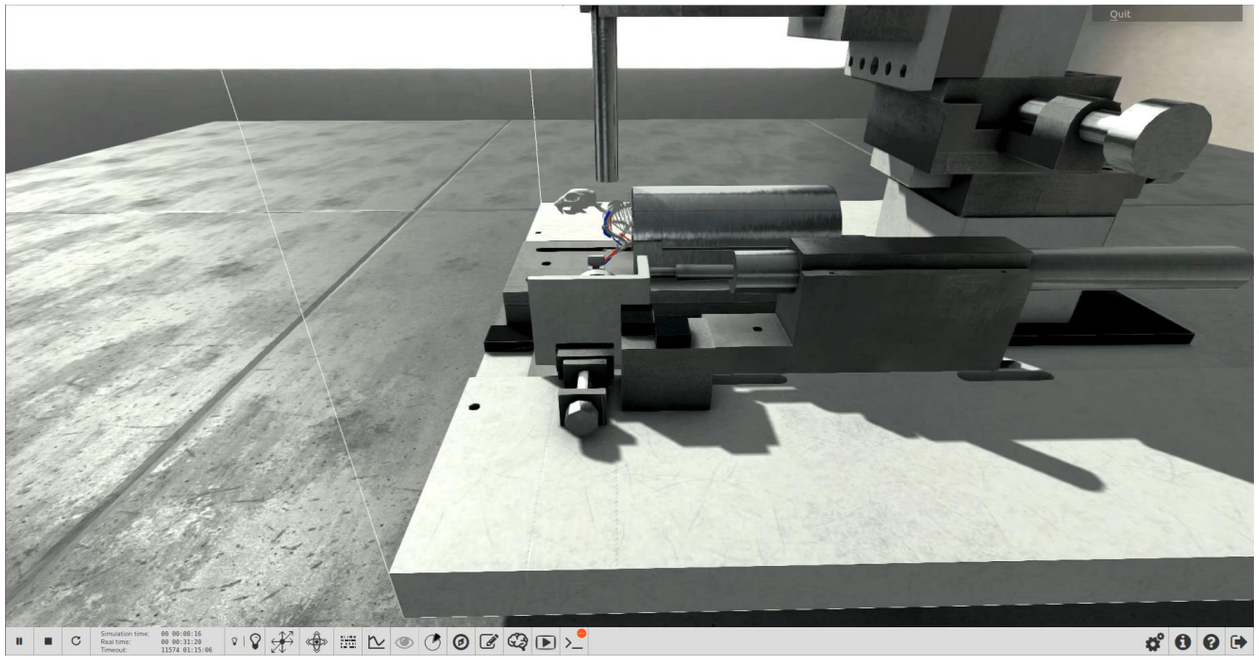


Robot-brain connection through a spinal cord model

If the robot has already muscle like-actuators, we do not need to employ the musculoskeletal simulation.

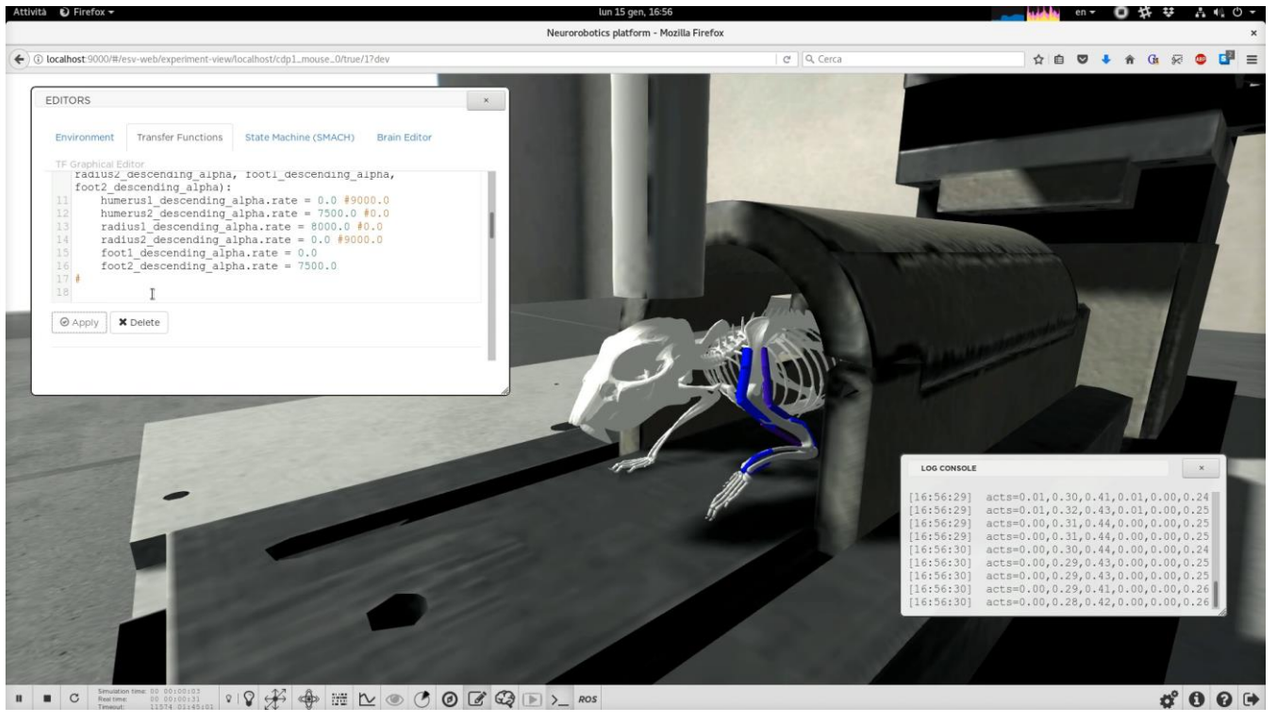
Spinal cord model as a general translation mechanism

The spinal cord model has been employed for (partially) reproducing real neuroscientific experiments.



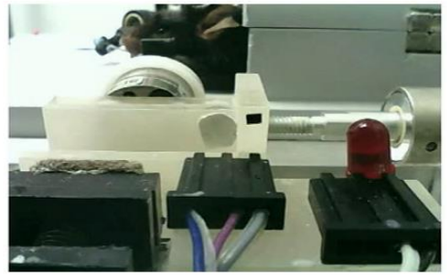
←Motor rehabilitation experiment

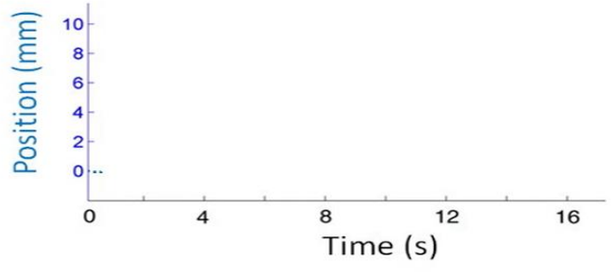
Reaching experiment→



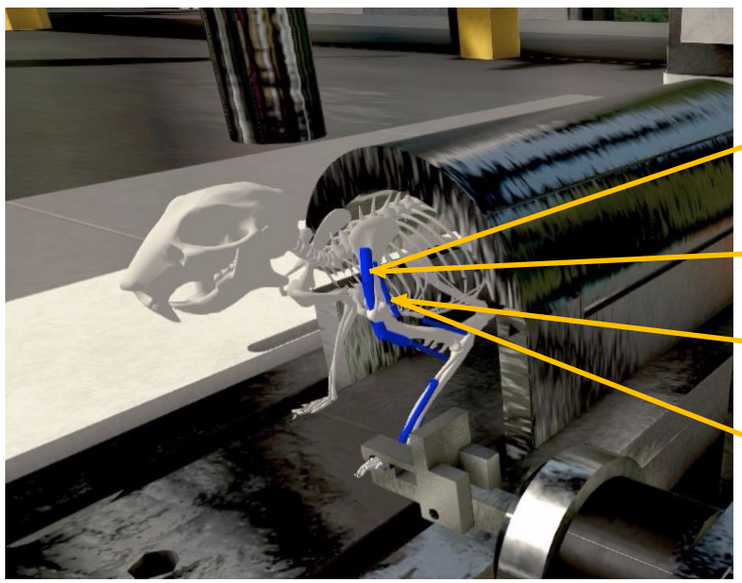
Post-stroke rehabilitation simulation

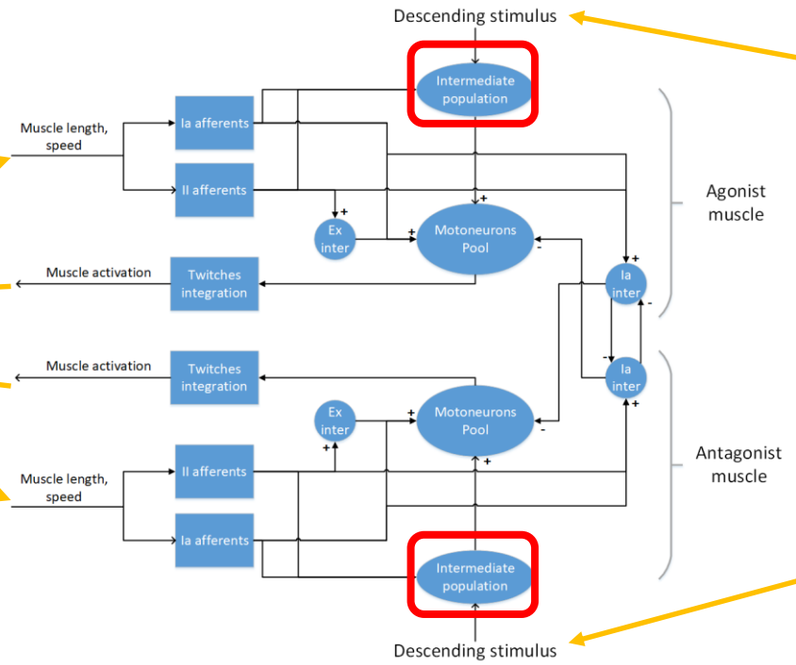
Learning of a forelimb pulling task, then study of motor task re-training in rodent model after induction of photothrombotic stroke with simultaneous intracranial recording.

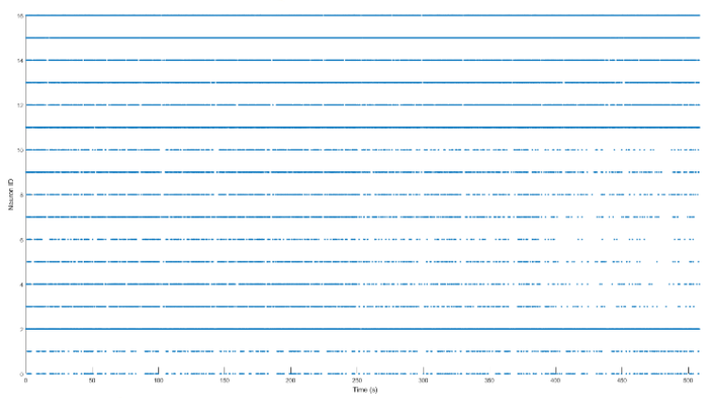




Reproduction *in silico*:







Recorded cortical activity

Neuromorphic computing resources

NEST:
- www.nest-simulator.org

SpiNNaker:
- spinnakermanchester.github.io
- apt.cs.manchester.ac.uk/projects/SpiNNaker

Other info (related):
- lorenzo.vannucci@santannapisa.it