

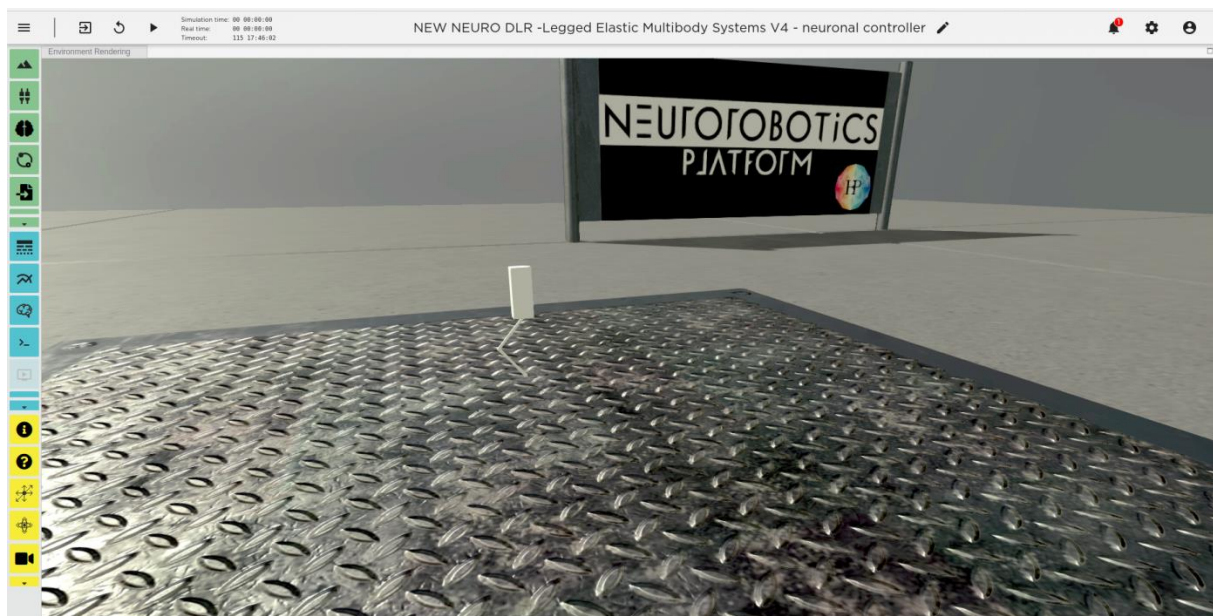
User Documentation: Neurocontrol

Introduction

Research suggests that the human brain transforms sensory input from the multi-dimensional joint space into a 1D synergy space along multiplicative weights to make sense of high-dimensional sensory data. Based on robotic control approaches and on previously known electrophysiological evidence, a plausible CNS model was found to scale the forces for different muscles during a periodic dynamic motion. It was implemented as a neurocontroller in the NRP and the use of it is described in this Documentation.

Setup

As an example, this experiment implemented the neurocontroller in a simple hopper, controlling its two joints. After cloning and launching the experiment, the following setup should show up:



TF Definitions

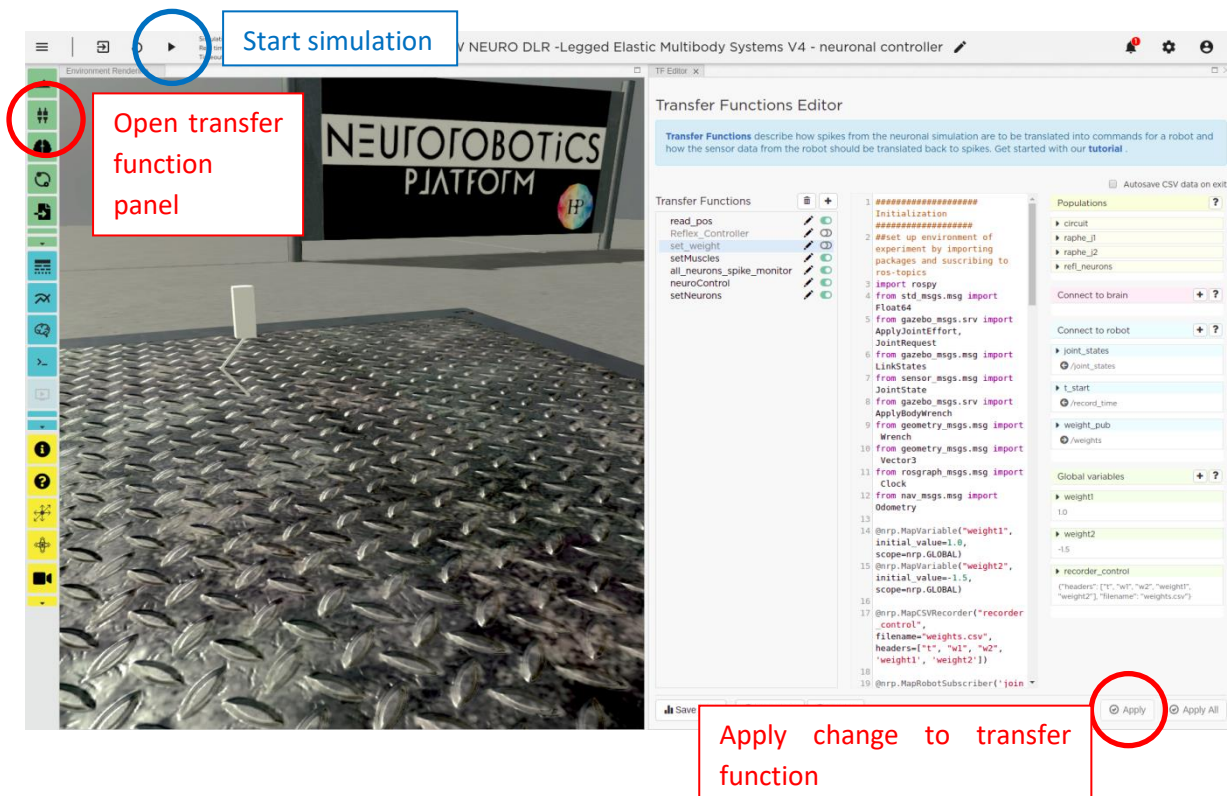
To open the transfer function, click the second green button from the top. A panel opens, in which the transfer functions defined in the .bibl-file can be found. In the Default Setup, the following functions that show up are:

- `setNeurons`: define the fire rate to map to the neuron populations in brain file
 - o Inputs:
 - `joint_states`: to read out joint positions to calculate fire rate for reflex neurons and raphe nuclei of each joint
(Default gazebo topic, `/gazebo/joint_states`, alternative: read out `joint effort`)
 - OPTIONAL: `link_states`: to read out coordinates of links, e.g. to look at body height
(Default gazebo topic: `/gazebo/link_states`)
 - o Outputs:
 - Fire rates for different neuron pools defined in brain file
(in NRP mapped through SpikeSources to populations defined in brain file)
- `setMuscles`: read out fire rate from neurons in brain file to
 - o Inputs:
 - Fire rates from neuron pools defined in brain file
(in NRP mapped through SpikeSink from populations defined in brain file)
 - o Outputs:
 - `apply_joint_effort` sending torques to the defined joints (`tau1`, `tau2`)
(ROS service: `/gazebo/apply_joint_effort`)
- `neuroControl`: combines `setNeurons` and `setMuscles` in one function
 - o Inputs:
 - see Inputs of `setNeurons` and `setMuscles`
 - o Outputs:
 - see Outputs of `setNeurons` and `setMuscles`
- `all_neurons_spike_monitor`: necessary to visualize spiking neurons in NRP
 - o Inputs/Outputs:
 - NONE
- `read_pos`: defines recorder to read out joint positions (useful for debugging)
 - o Inputs:
 - `joint_states`: to read out joint positions to calculate torques in joints
(Default gazebo topic, `/gazebo/joint_states`, alternative: read out `joint effort`)
 - `link_states`: to read out coordinates of links, e.g. to look at body height
(Default gazebo topic: `/gazebo/link_states`)
 - o Outputs:
 - NONE

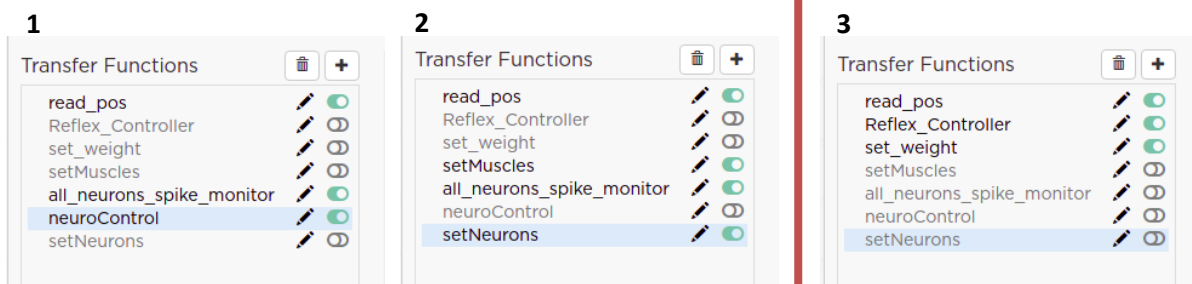
- Reflex_Controller: defines how the control signal is applied to robot
 - o Inputs:
 - weights: to tune control signal for different joints
(calculated in set_weight and published through ROS)
 - joint_states: to read out joint positions to calculate torques in joints
(Default gazebo topic, /gazebo/joint_states, alternative: read out joint effort)
 - OPTIONAL: link_states: to read out coordinates of links, e.g. to look at body height
(Default gazebo topic: /gazebo/link_states)
 - OPTIONAL: t_start: start time to start recording when saving .csv –file
(defined in set_record_time and published through ROS)
 - o Outputs:
 - apply_joint_effort sending torques to the defined joints (tau1, tau2)
(ROS service: /gazebo/apply_joint_effort)

- set_weight: define how the weights are adjusted (Default: 3 → Oja Rule)
 - o Inputs:
 - joint_states: to read out joint positions to calculate weights with gradient descent method or Oja rule.
(Default gazebo topic, /gazebo/joint_states)
 - OPTIONAL: t_start: start time to start recording when saving .csv –file
(defined in set_record_time and published through ROS)
 - o Outputs:
 - weights: needed to tune control signal for different joints
(published through ROS as topic /weights)

Use



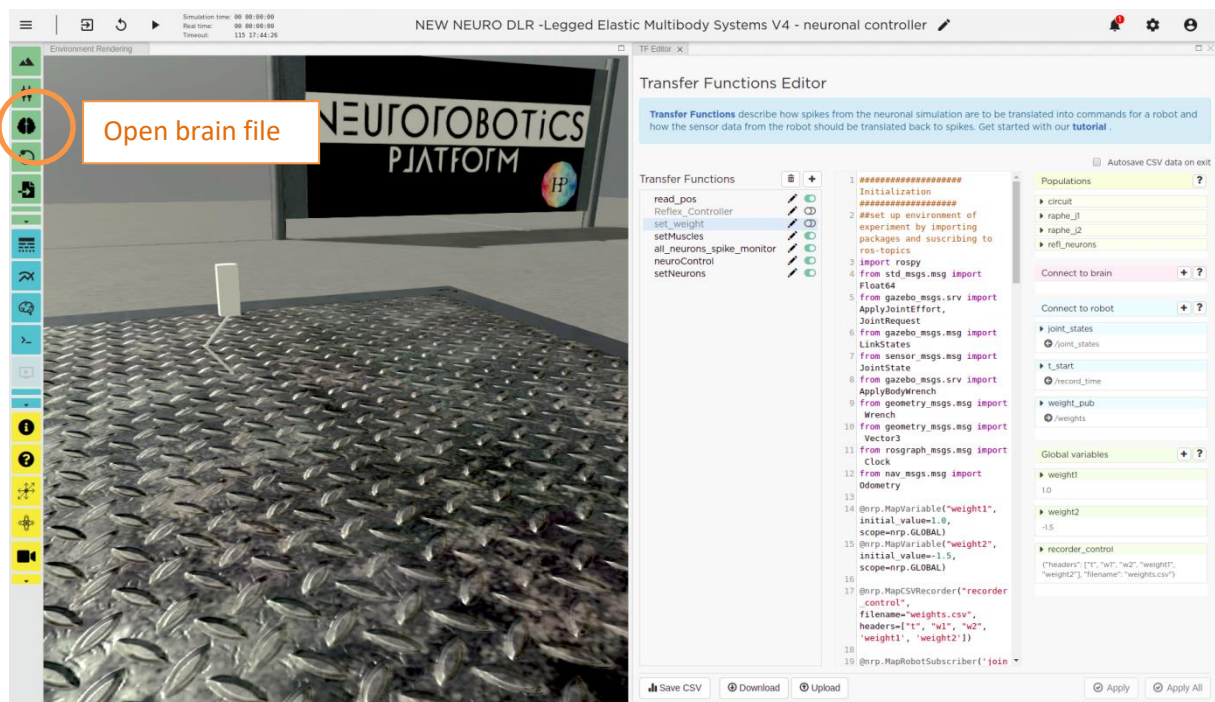
In this experiment, not all transfer functions need to be used at the same time. Applying the functions as seen below allows running different controller in the same setup:



The two left options (1 and 2) are equivalent in application, since the `neuroControl` combines the functions of `setNeurons` and `setMuscles`. However, when tuning the controller for a new system, it might be easier to look at both components separately.

The combination of transfer functions shown on the right (3) allows the application of an equivalently working robotic controller. For more information on that, see the *User and Tech Documentation* of the NRP experiment about Mode Extraction on the Myoarm. This manual explains mainly how to use the neurocontroller. However, since the Reflex controller has been helpful to tune parameters for the neuro control and is therefore included in this experiment. If it is not needed, the `Reflex_Controller` and `set_weight` function can be deleted from the .bibi-file so they do not show up in the NRP panel.

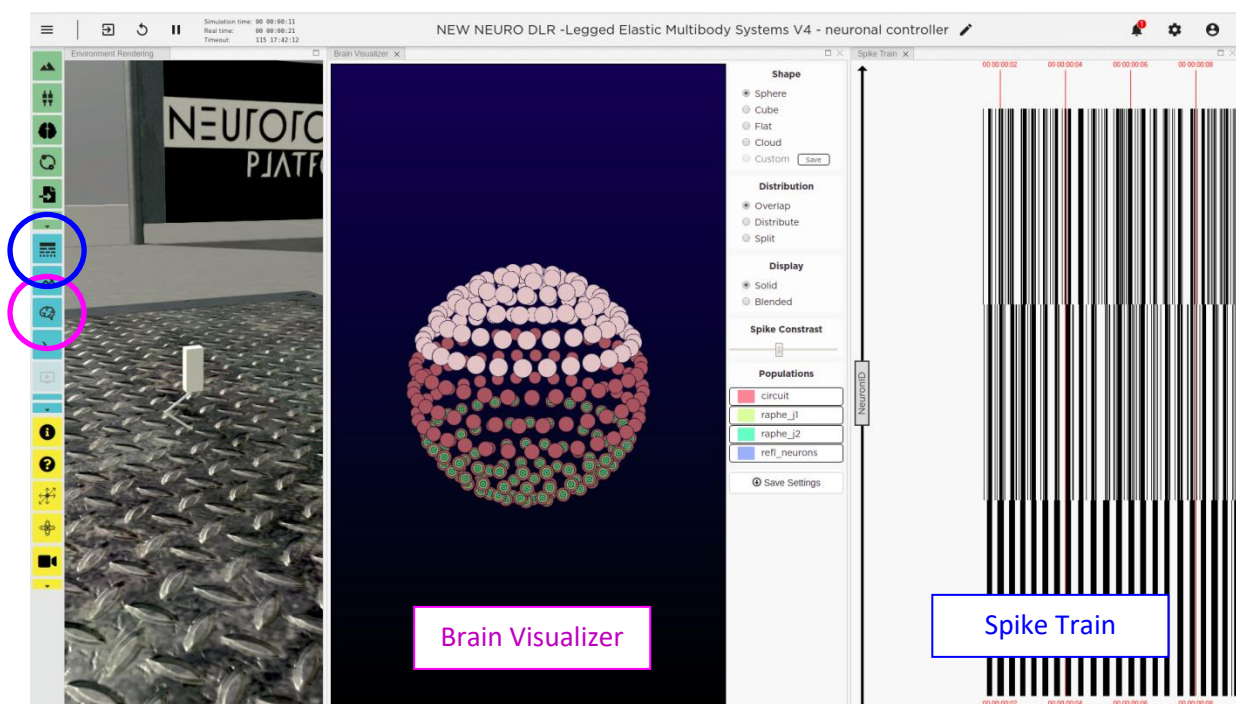
To check or change the setup of the brain file, press the third green panel on the left (see below):



In the Default setup, there are three neuron populations, each with 100 neurons: 2 Raphe Nuclei populations, one reflex population. For the functioning of those neuron pools and control principle in general, please refer to the *Tech Documentation*.

After turning on the appropriate functions for the neuro control (Option 1 or 2 above), press the blue “Play” button on the top bar. This will let the hopper fall from its initial position and the neurons will adapt the weighting of the control signal automatically.

If you want to check the spiking of the neuron populations in the NRP open the Brain Visualizer or the Spike Train.



IBA

Instead of using the transfer functions, it is also possible to use the IBA, which already implement the function `neurocontrol.py` and the brain file. The inputs and outputs of the function remain the same as stated in the section “TF Definitions”.

When starting the transfer function in this way, the hopper will fall from its initial position and start jumping, driven by the spiking neural network.

For more details on the control strategy, please refer to the *Tech Documentation*.