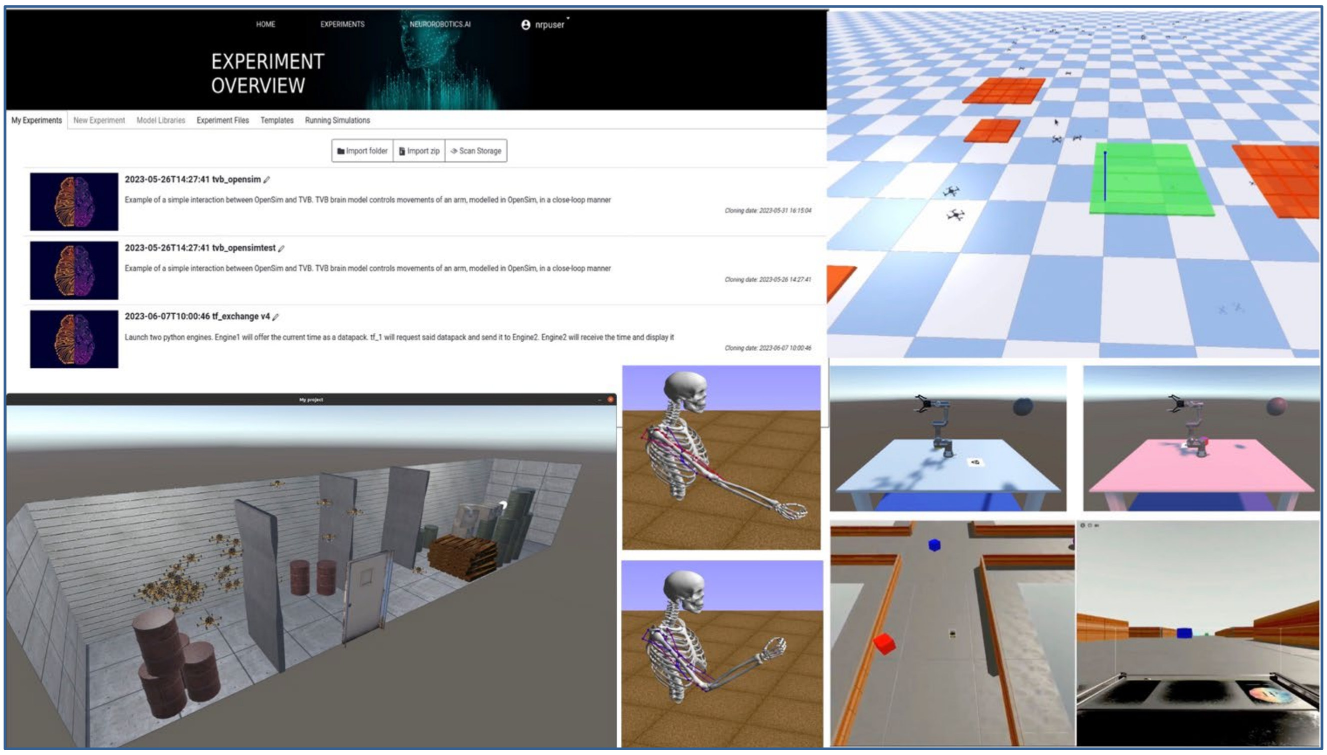
EBRAINS closed-loop AI and robotics workflows (D5.9 - SGA3)



2023-05-26T14:27:41 tvb_opensim

Example of a simple interaction between OpenSim and TVB. TVB brain model controls movements of an arm, modelled in OpenSim, in a close-loop manner

Cloning date: 2023-05-31 16:15:04

2023-05-26T14:27:41 tvb_opensimtest

Example of a simple interaction between OpenSim and TVB. TVB brain model controls movements of an arm, modelled in OpenSim, in a close-loop manner

Cloning date: 2023-05-26 14:27:41

2023-06-07T10:00:46 tf_exchange v4

Launch two python engines. Engine1 will offer the current time as a datapack. tf_1 will request said datapack and send it to Engine2. Engine2 will receive the time and display it

Cloning date: 2023-06-07 10:00:46

Figure 1: Neurorobotics Platform v4.1, new front end and use cases

With the release of NRP 4.1, the Neurorobotics Platform is now a highly modular and versatile simulation framework with new features and a new front end (Section 2) that support the deployment of concurrent NRP instances for learning and optimisation, the generation of randomised synthetic data, and the reuse and integration of neurocomputational models (Section 3).

Project Number:	945539	Project Title:	HBP SGA3

Document Title:	EBRAINS closed-loop AI and robotics workflows
Document Filename:	D5.9 (D56) SGA3 M36 SUBMITTED 230718.docx
Deliverable Number:	SGA3 D5.9 (D56)
Deliverable Type:	Other
Dissemination Level:	PU
Planned Delivery Date:	SGA3 M36 / 31 Mar 2023
Actual Delivery Date:	SGA3 M40 / 18 Jul 2023
Author(s):	Fabrice MORIN, TUM (P56) Francesca CAVALLARO, TUM (P56)
Compiled by:	Francesca CAVALLARO, TUM (P56)
Contributor(s):	Eloy RETAMINO, UGR (P66), contributed to Section 2.1.2 Sandro WEBER, TUM (P56), contributed to Section 2.1.1 Viktor VOROBEV, TUM (P56), contributed to Section 3.3 Ugo ALBANESE, SSSA (P49), contributed to Section 3.2 Krzysztof LEBIODA, TUM (P56), contributed to Section 2.1.3 and 3.1 Vahid ZOLFAGHARI, TUM (P56), contributed to Section 3.1 Deniz ERGENE, TUM (P56), contributed to Section 3.1 Bernardo MARTINEZ, UGR (P66), contributed to Section 3.2
WP QC Review:	Evita MAILLI, ATHENA (P133)
WP Leader / Deputy Leader Sign Off:	Fabrice MORIN, TUM (P46)
T7.4 QC Review:	Martin TELEFONT, Annemieke MICHELS, EBRAINS (P1)
Description in GA:	Neurorobotics Platform release with improved functionality, extended content, and updated inventory of related closed-loop models, tools, and services prepared for, or released through the EBRAINS portal.
Abstract:	This document provides a high-level overview of the evolution of the Neurorobotics Platform (NRP) in SGA3 and describes the main features of the latest release (version 4.1), which provides our users with necessary updates in parts of the software stack in terms of both usability and functionality. Examples are included that illustrate how multiple instances of an NRP-based simulation can now be gracefully executed to support various AI, robotics and neuroscience workflows that either require or leverage gradient-based learning approaches (e.g. Reinforcement Learning) or gradient-free optimisation methods (e.g. genetic algorithms). Additionally, multiple computational models coming from various HBP collaborators were integrated with NRP v4.1 and are also herein presented. Finally, the integration of the NRP with NEST-Desktop is described, showcasing how these two tools developed within the HBP can be used in the classroom to provide students with a better understanding of the relationship between neural network structure and function when teaching computational neuroscience.

Keywords:	Neurorobotics, embodied cognition, simulation, closed-loop experiments, neuroscience, embodied AI, sensorimotor loop, spiking neural networks
Target Users/Readers:	Computational neuroscience community, computer scientists, HBP Consortium members, embodied AI researchers, roboticists, general public, neuroscientific community, students.

Table of Contents

Table of Tables

Table of Figures

1. Introduction

The **Neurorobotics Platform** (**NRP**) was developed in the context of the Human Brain Project (HBP) to provide neuroscientists with a set of software tools to address the need for embodiment in simulation, i.e. connecting brain models with bodies and taking into account the interactions of the agent with a physically realistic environment when simulating brain processes. The rationale behind this approach is that realistic modelling of proprioceptive and exteroceptive sensory streams is essential to simulating a naturalistic and relevant brain activity. Since the inception of the HBP in 2013, the NRP strived to address this requirement, and at the beginning of 2020, a version was released (version 3.0) that provided a robust way to do so – at least in a limited set of environments compatible with the Gazebo simulator.

Work was thus concurrently started to increase the range of use cases that the NRP could support. It was decided to create a brand-new version of the NRP based on a fundamentally different software architecture (**NRP v4.x**) to engage new users through high modularity of the platform, while maintaining and improving the so-called "legacy" version (***NRP v3.x***) for existing current users. Interested readers will find discussion of the need for the evolution of the NRP and the corresponding roadmap in the previously released SGA3 Deliverables D5.1 (D48) and D5.6 (D53), which also provide a description of the state of the NRP in September 2021.

The present document specifically provides a high-level view of the evolution of the NRP after September 2021. It details the technical developments that were undertaken in support of the vision laid out in the previous Deliverables, vision that led to the development and release in June 2023 of the latest (and last version in the context of the HBP) of the NRP. This latest version, **NRP v4.1**, possesses the modular software architecture in the back end that was introduced in previous Deliverables, a new REACT front end to specifically support the aforementioned modularity, and usability features such as a client for the NRP core component that facilitates distribution and integration into learning experiments.

It should be noted that the present Deliverable does not cover some activities related to the NRP software development. Indeed, support activities for users of the current online NRP version (v3.2.x) were not discontinued[1]. Improvements and bug fixes were introduced in May 2023 through release 3.2.2 of the legacy NRP to improve the user experience with the online version of the platform, which will remain available until the end of the HBP. Nevertheless, they represent a rather incremental, as opposed to fundamental, evolution, and as such the interested reader is referred to the extensive online documentation available[2].

The present document instead delves into the presentation of the aforementioned technical features of NRP v4.1 (Section 2) and presents showcases of the new possibilities enabled by said features for the users of the Platform (Section 3). These showcases demonstrate that the new NRP can now cover a much wider range of applications and use cases than the legacy version. Examples are provided that include reuse and integration of neurocomputational models, deployment of concurrent NRP instances for learning and optimisation (including on the EBRAINS compute infrastructure, namely the Jusuf supercomputer in Jülich, Germany), and educational use.

Finally, the last section (Section 4) provides a look back at the most significant contributions of the Neurorobotics Platform to the Human Brain Project, as well as some perspectives and possible pathways towards a future evolution of the NRP outside of the scope of the Project that supported its development for the past decade, especially in terms of embodied AI for medical technology.

[1] Technical support for NRP users is offered via our dedicated NRP Forum and via the HLST contact page.
[2] https://neurorobotics.net/Documentation/legacy/index.html

2. New significant features of NRP v4.1

2.1 Overview

The extensive efforts invested in the development of the new NRP v4.x architecture allowed us to achieve the following:

- Modularity and parallel execution across engines (including deep NNs, spiking NNs, and any type of standard controller)
- Implementation of distributed optimization/learning algorithms
- Capacity to integrate new tools in the NRP ecosystem
- Easy distribution on multiple machines
- Avoidance of dependency conflicts
- Each engine executes at its own frequency

Table 1 highlights the main advantages of NRP v4.x hub and spoke architecture over NRP v3.x centralised architecture.

Table 1: NRP v4.x and NRP v3.x architecture comparison

Feature	v3.x (centralised)	v4.x (hub and spoke)
Modularity and Simulator Support	Only Gazebo (world) and spiking neural-network simulators.	• Full modularity. Any user-defined simulator can be implemented. • Supporting new simulators is easy. • Engines can be any Python script processing Datapacks (NRP units of data exchange).
Distribution	• It is not possible to distribute simulators. • NEST own MPI support employed in Feldotto et al., 2022[3].	• Engines and NRP Core communicate via network. • Built-in support for distribution (e.g., using docker container as distribution unit).
Maintainability and Extensibility	Any change to simulator spreads to other components, thereby making integration and updates expensive.	Engines and NRP Core are decoupled. As long as their interface is not impacted, changes remain local, and barriers to integration and update are eliminated.

2.1.1 *New Front End*

Given the outdated framework dependencies in the old NRP front end codebase, which resulted in security risks, and to support the increased modularity of NRP v4.x new architecture, a new front end codebase foundation centred on **ReactJS** was developed. Indeed, ReactJS allows a separation of progressive GUI code (ReactJS) that integrates modular (external) tool dependencies into GUI components and background services, thereby avoiding monolithic interdependent code in favour of more flexible building blocks (see SGA3 Deliverable D5.6 for more details).

[3] Feldotto, et al., (2022) *Deploying and Optimizing Embodied Simulations of Large-Scale Spiking Neural Networks on HPC Infrastructure*. Front. Neuroinform. 16:884180. doi: 10.3389/fninf.2022.884180 PLUSID: P3266

After establishing this new front-end codebase, efforts focused on the integration of different features and requirements of NRP v4.x, while the legacy NRP v3.x keeps operating on the old frontend codebase.

Below, we present some of the major changes implemented through such development efforts.

Service communication between the front end and (in its extension) NRP proxy was adjusted to match the new NRP v4.x back end. For *asynchronous communication*, MQTT was added to the front end and the bare client was extended to provide all the necessary publish/subscribe behaviour, so that the modular simulation tools can now work independently of each other and without interference when accessing the same data channels.

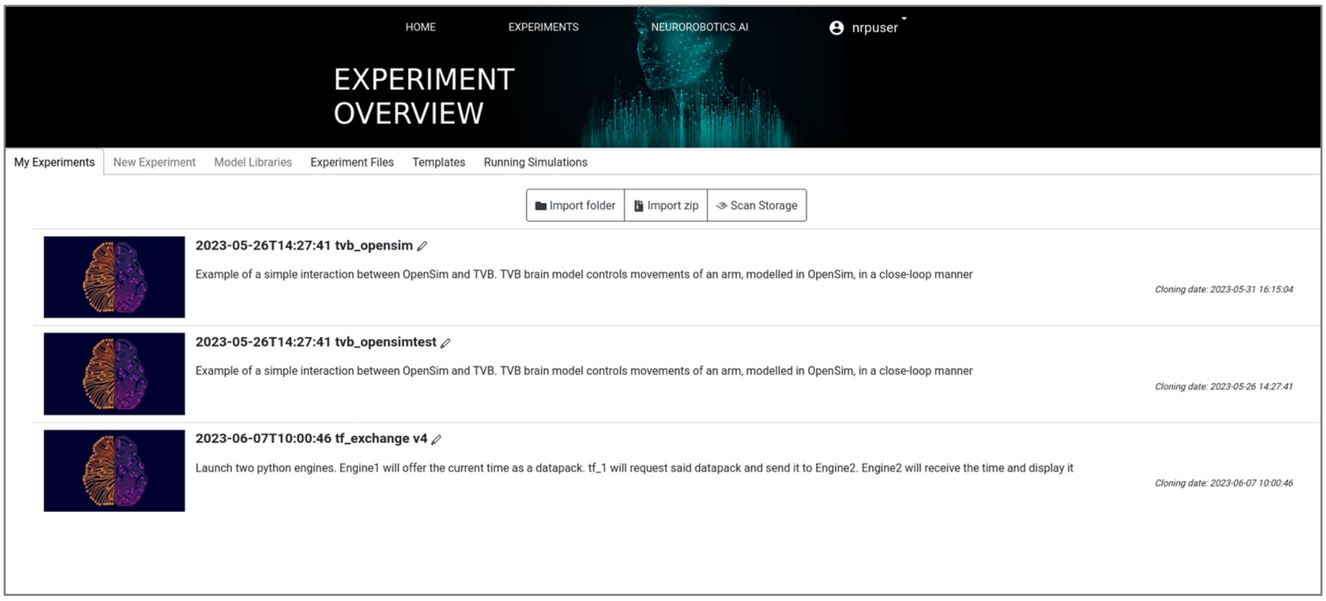
For the ***Data Visualisation and Plotting tool***, an adapter for MQTT communications, was added (similar to what has been implemented in the past for ROS topic communication) to allow plotting of data from MQTT channels using the same implementation and mechanisms. This means that the plotting tool can also be easily adapted to other communication channels in the future.

A remote display server based on Xpra was added to the back end, providing the front end with a simulation tool for remote application and desktop interaction. This will serve as a general solution when integrating NRP with simulation tools that must be run natively on the simulating machine (for example Gazebo Desktop GUI).

The ***integration with the EBRAINS Collab and Bucket file storage infrastructure*** required extensive efforts and, in some cases, even led to structural changes to the NRP proxy server and its behaviour. For instance, integration with the old Collab, preceding EBRAINS, relied on matching implementation and interchangeable local mode behaviour aligned with the dual working mode of the NRP - local/offline and online storage/execution of experiments. Given the different endpoints and behaviour of the new Collab instead, some of the overall proxy structure of the NRP had to be adjusted in order to maintain consistency in the code between both working modes.

Finally, to improve navigation and usability the front end now displays a more suitable landing/home page with links to the news section from www.neurorobotics.net as well as quick-access links to the last experiments opened (Figure 2). The *info & error* handling mechanisms and notification pop-ups were also improved.

Users can find all the front-end-related code at https://bitbucket.org/hbpneurorobotics/nrp-frontend/src/master/.



2023-05-26T14:27:41 tvb_opensim

Example of a simple interaction between OpenSim and TVB. TVB brain model controls movements of an arm, modelled in OpenSim, in a close-loop manner

Cloning date: 2023-05-31 16:15:04

2023-05-26T14:27:41 tvb_opensimtest

Example of a simple interaction between OpenSim and TVB. TVB brain model controls movements of an arm, modelled in OpenSim, in a close-loop manner

Cloning date: 2023-05-26 14:27:41

2023-06-07T10:00:46 tf_exchange v4

Launch two python engines. Engine1 will offer the current time as a datapack. tf_1 will request said datapack and send it to Engine2. Engine2 will receive the time and display it

Cloning date: 2023-06-07 10:00:46

Figure 2: Screenshot of the new NRP front end landing page

2.1.2 *Capacity for soft real-time simulations*

With the release of NRP v4.1, the NRP can now provide its users with tools for the execution of modular simulations in ***soft real-time***. This section describes such tools and their potential application to Sim2Real transfer in robotics.

To simulate experiments in the NRP, a set of Engines runs instances of a particular simulation and exchanges data through a central component, the NRP Core[4]. Until NRP v4.x, experiments were always run in a fully *synchronous* mode, in which the NRP Core, by making RPC (remote procedure calls) and blocking calls, requested Engines to advance their simulations, and managed data exchanges. Thereby, the NRP is endowed with deterministic execution and reproducibility, which are fundamental features for scientific experiments of embodied simulation. This setup is also suitable for implementing modular machine learning experiments which can be distributed and run faster than real-time in a deterministic manner (i.e. without introducing noise and uncertainty that would otherwise be caused when using *asynchronous communications* between experiment components).

For cases where the final goal of an experiment is to interact with *real-time* systems, such as robots or cyber-physical systems in general, ***NRP v4.1 offers now the further possibility to execute experiments in real-time and with asynchronous communication via a soft real-time component called EventLoop***. The Event Loop (https://en.wikipedia.org/wiki/Event_loop) provides a generic mechanism to process asynchronous events in real-time and stream others in response by running a loop at a fixed frequency, which is differently applied to Engines and to the central component but has the same structure in both cases.

As illustrated in Figure 3, each Event Loop operates as follows:

1) process incoming Events (red boxes), wherein input events are data sent by other Engines or the central component
2) execute either computations for advancing the Engine simulation or a Computational Graph in the case of the central component, in order to generate output events
3) stream out said output events (blue boxes)

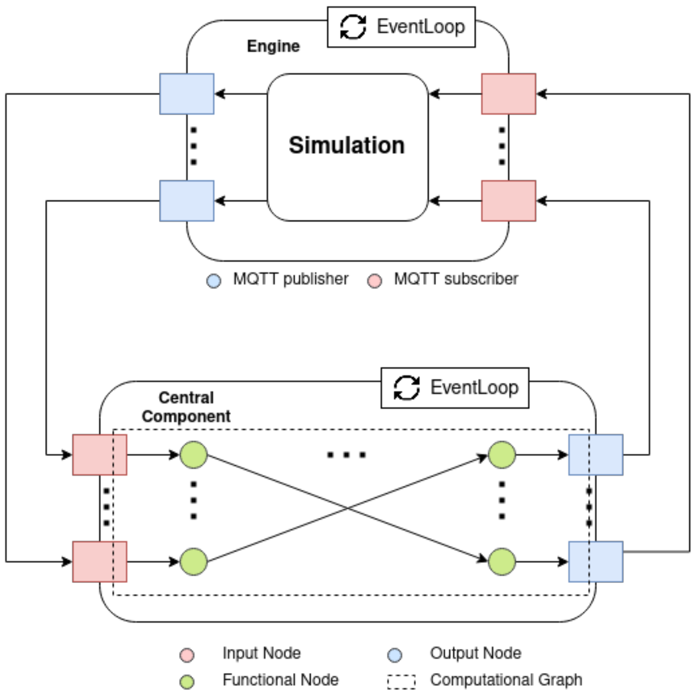


Figure 3: Experiment executed in soft real time with asynchronous communications

Engines and the central component run independently from each other, and each is managed by its own EventLoop. Both Engines and the central component use MQTT to exchange data.

[4] https://neurorobotics.net/Documentation/latest/nrp-core/index.html

The **Computational Graph** is a new component introduced in NRP v4.1 release. It implements a graph structure with three kinds of nodes: Input Nodes, Functional Nodes and Output nodes. External messages are injected into the graph via Input Nodes using different communication protocols. Currently supported protocols include ROS topics, MQTT topics and a SPINNAKER board through a raw TCP connection. Functional Nodes receive messages from Input Nodes or other Functional Nodes, pass them as arguments to a function which contains the node computation. The function output(s) can be connected to other nodes. Functional Nodes can run both Python and pre-compiled C++ functions. Output nodes stream received messages using any of the supported communication protocols.

The implementation of soft real-time execution in the NRP v4.1 has been designed so that switching from fully synchronous experiments to asynchronous ones whose components run independently in real time is achieved by simply applying changes to the experiment configuration. This is a significant difference between NRP v4.1 and other popular frameworks in robotics such as ROS (Robot Operating Systems), where a substantial effort and proper technical knowledge would be required to implement such sort of switching, or other machine learning frameworks that do not consider communication between components or real-time execution. This easy change to the experiment configuration is a key feature in the case of a Sim2Real transfer workflow. In the NRP v4.1 this workflow now encompasses the following steps:

1) First, the user designs the experiment **using the fully synchronous mode** with one or more Engines simulating an actual system of interest and one or more Engines to ultimately interact with it, acting as controllers, monitors or implementing some sort of information processing mechanism in general. Engines then exchange data through a Computational Graph and are managed by the central component, NRP Core.

2) After the user is satisfied with the behaviour of the simulated, fully synchronous experiment, they will **switch to the asynchronous mode**, wherein all Engines and the central component are configured to run in real-time and communicate asynchronously using independent Event Loops. In this step the user can observe the effects of *asynchronicity* in the communication and therefore runtime non-determinism while still staying in the simulated environment. Required changes or tuning can be performed here.

3) Once Engines are ready for connecting to the actual system, connections in the Computational Graph to/from its simulated counterpart are replaced. Depending on the interface of this actual system, this step will probably entail changes in the Computational Graph code to redirect output messages to the appropriate endpoints using ROS or MQTT topics.

This Sim2Real transfer workflow is currently being used by the Applied Computational Neuroscience (ACN) Group at University of Granada (UGR) in neurorobotics learning experiments. The main research hypothesis is that Spiking Neural Network (SNN) based controllers should be particularly well suited to facilitate the Sim2Real transfer, given the inherent adaptability of the models used therein and their tight integration with online learning algorithms. The main research interest of this effort is to analyse the process of training spiking cerebellar models to perform a particular control task in a simulated, fully synchronised experiment, and to deploy the trained controller in a real robot. The aforementioned tools for real-time execution and Sim2Real transfer are being extremely useful to test this hypothesis.

2.1.3 Concurrent execution of multiple simulation instances

Concurrency, the ability to execute multiple and alike simulations *concurrently* (in parallel) is crucial when working with modern machine learning algorithms, where models often require hundreds of thousands of samples to learn from. While NRP v4.0 already offered the possibility to have multiple simulations running in parallel as separate engines, ***NRP v4.1 now enables users to vectorise NRP Core itself***, i.e. to parallelise and synchronise multiple instances of NRP Core. This is thanks to a specific component, the NRP Python Client[5] (referred to as Client), which allows a custom Python script (later called the Main Script) to spawn and orchestrate NRP Core instances

[5] https://neurorobotics.net/Documentation/versions/4.x-1614/nrp-core/page_python_client.html

(processes), as well as gather data (observations) from and send commands (actions) to its engines (Figure 4).

Several ways of achieving concurrency are therefore possible within the NRP Core toolbox of the new NRP v4.1:

1) parallelisation, and possibly vectorisation, at the level of NRP Core (by using the Client)
2) multiple similar engines working as part of a single NRP Core instance
3) multiple agents simulated within a single engine
4) a combination of the above approaches

When choosing (1), NRP Core vectorisation, the Main Script spawns multiple instances of NRP Core in separate processes, and communicates with them in a non-blocking manner using the Clients. Rudimentary capabilities for non-blocking (***asynchronous***) communication are built into the Client. However, when combining such capabilities with vectorisation tools, the workflow necessary for handling one or multiple instances of NRP Core becomes almost the same, without the need for writing loops and extra code for data aggregation. An example of such a vectorisation tool is provided by frameworks such as Stable Baselines3 (https://stable-baselines3.readthedocs.io/en/master/), a popular library of Reinforcement Learning algorithms.

The ability to vectorise NRP Core and execute simulation steps in asynchronous manner is what ushers in the potential for massive parallelisation of simulations.

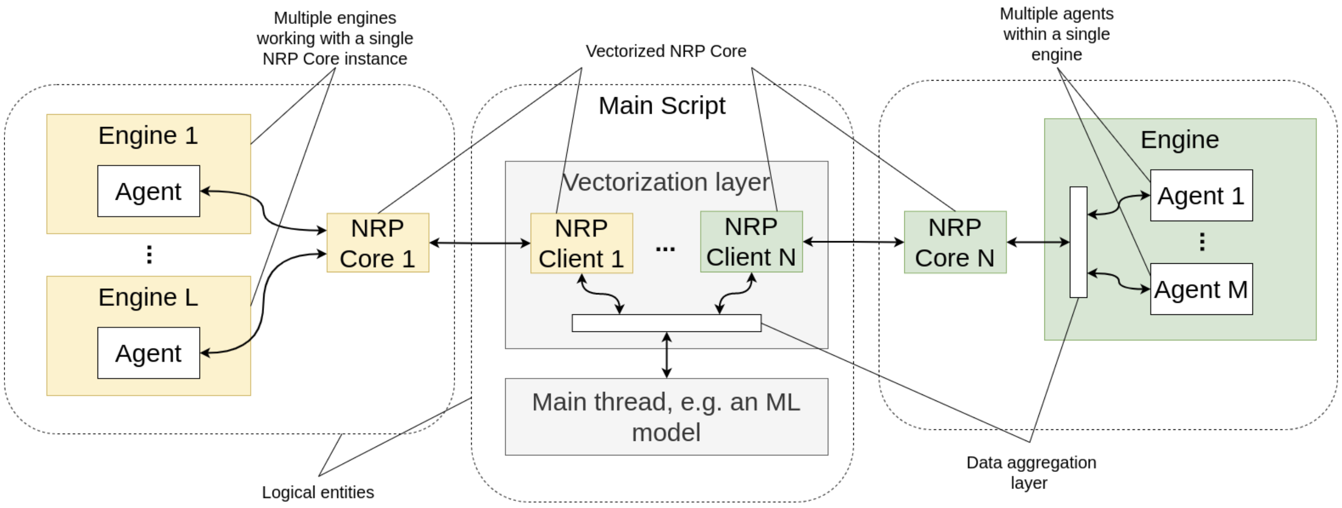


Figure 4: Concurrency at different levels in the NRP

Multiple instances of NRP Core acting as workers for the main thread of the Main Script; multiple engines acting as services to a single NRP Core (left), and multiple agents within a single simulation (right). The Main Script, NRP Cores, and Engines, are separate processes, possibly running on different computers.

In the implementation of concurrency, the ability to ***containerise and distribute the simulations*** is as important as vectorisation. This can also be achieved at different levels with NRP Core – either by distributing containerised engines with the help of a special mechanism called NRP Docker Launcher, or by distributing the whole NRP Core instances (NRP Core together with its engines). The latter is possible by using distribution tools provided by the well-known docker compose (https://docs.docker.com/compose/) and docker swarm tools (https://docs.docker.com/engine/swarm/), which are now integrated into the NRP Core ecosystem.

3. Examples of applications enabled by the architecture of NRP v4.1

3.1 Learning and optimisation with distributed concurrent NRP instances

Concurrent Simulation of Multiple Drone Navigation in A Unity Environment

To show how NRP Core can be *vectorised* we performed an experiment in which multiple agents, **drones,** are simulated concurrently in an environment created with **Unity** game engine[6]. In this case, Unity acts as a service (Engine) to NRP Core thanks to the ***NRP Grpc Unity Plugin***, while NRP Core Client is embedded in a wrapper class that conforms to the interface required by the algorithms implemented in **Stable Baselines3**[7] (SB3), Multiple instances of NRP Core are vectorised using the Client and tools provided by SB3.

Only high-level actions (move forward, left, right, up, etc.) are considered, and no low-level flight control of the drones is implemented. Observations consist of the drone position in the global 3d space. ***The goal of this experiment is to generate a policy that will allow the drones to navigate and reach a specific target in a narrow corridor*** (Figure 5). Learning of the policy is performed in the Main Script using the PPO (Proximal Policy Optimisation) algorithm implementation provided by SB3 and it happens on-line, which means that the algorithm keeps interacting with the environment during the process. At each simulation step, the wrapper class distributes the actions produced by the current policy to the proper NRP Core instances, executes simulation steps in a non-blocking manner, and aggregates the resulting observations, which are later returned to the Main Script. The NRP Core asynchronous communication capabilities provided by the Client are in use here, and it is worth mentioning that the user can specify the number of NRP Core instances, as well as the number of drones simulated by each instance.

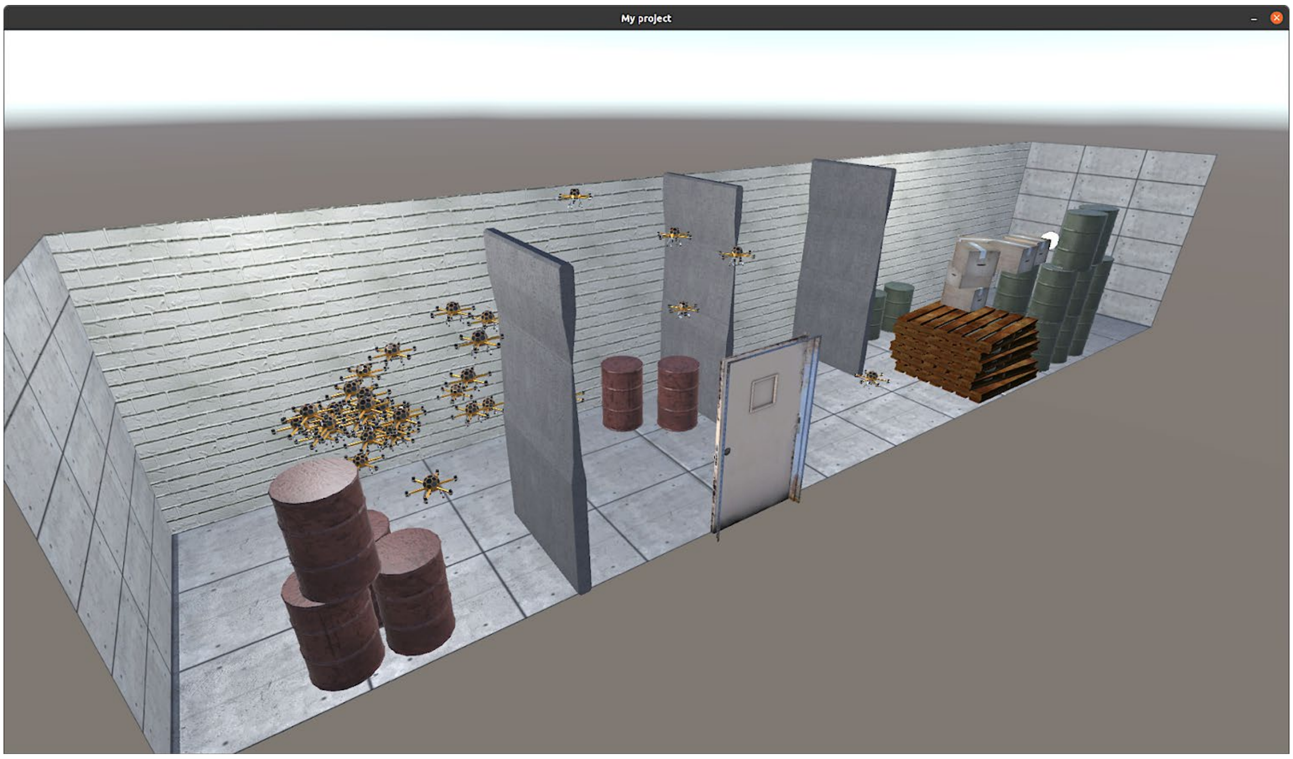


Figure 5: Learning of policy for navigation to target

Target (white circle); navigation simulated using PPO algorithm (Stable Baselines3) with multiple agents (drones) in a single Unity environment.

[6] https://unity.com/

[7] https://stable-baselines3.readthedocs.io/en/master/

Evolutionary Optimisation of a Quadrotor Drone Controller

Another, more complex, experiment explored the ***evolutionary optimisation*** of a quadrotor drone controller. In this case, NRP Core was employed to orchestrate multiple heterogeneous components inside specific engines. The Engines are a **PyBullet** based drone simulation, a hierarchical PID (Proportional-Integral-Derivative Controller)-based controller for low-level control, and a spiking neural network implemented in the NEST simulator for high-level swarm control. Evolutionary optimisation on an HPC infrastructure was realised via the **Learning to Learn (L2L) framework**. The low level PID controller was tuned on various simple tasks. Distribution over multiple nodes of an HPC system allows for fast exploration and convergence in a large parameter space. Additionally, the NRP and L2L allow for easy definition of custom tasks and optimisation algorithms. ***The resulting controller is capable of performing manoeuvres that go beyond the complexity of the simple tasks it was tuned with***. The spiking high-level controller was optimised to solve ***swarm tasks*** using neuro-evolution (Figure 6).

The modular nature of the NRP v4.x facilitates the combination of multiple levels of control into a single working solution. This makes it straightforward to interface the higher and lower-level controllers, which internally use different frameworks. Both controllers and the drone simulator may be swapped with other solutions, if the user wishes so, by simply changing the transceiver functions to ensure their compatibility.

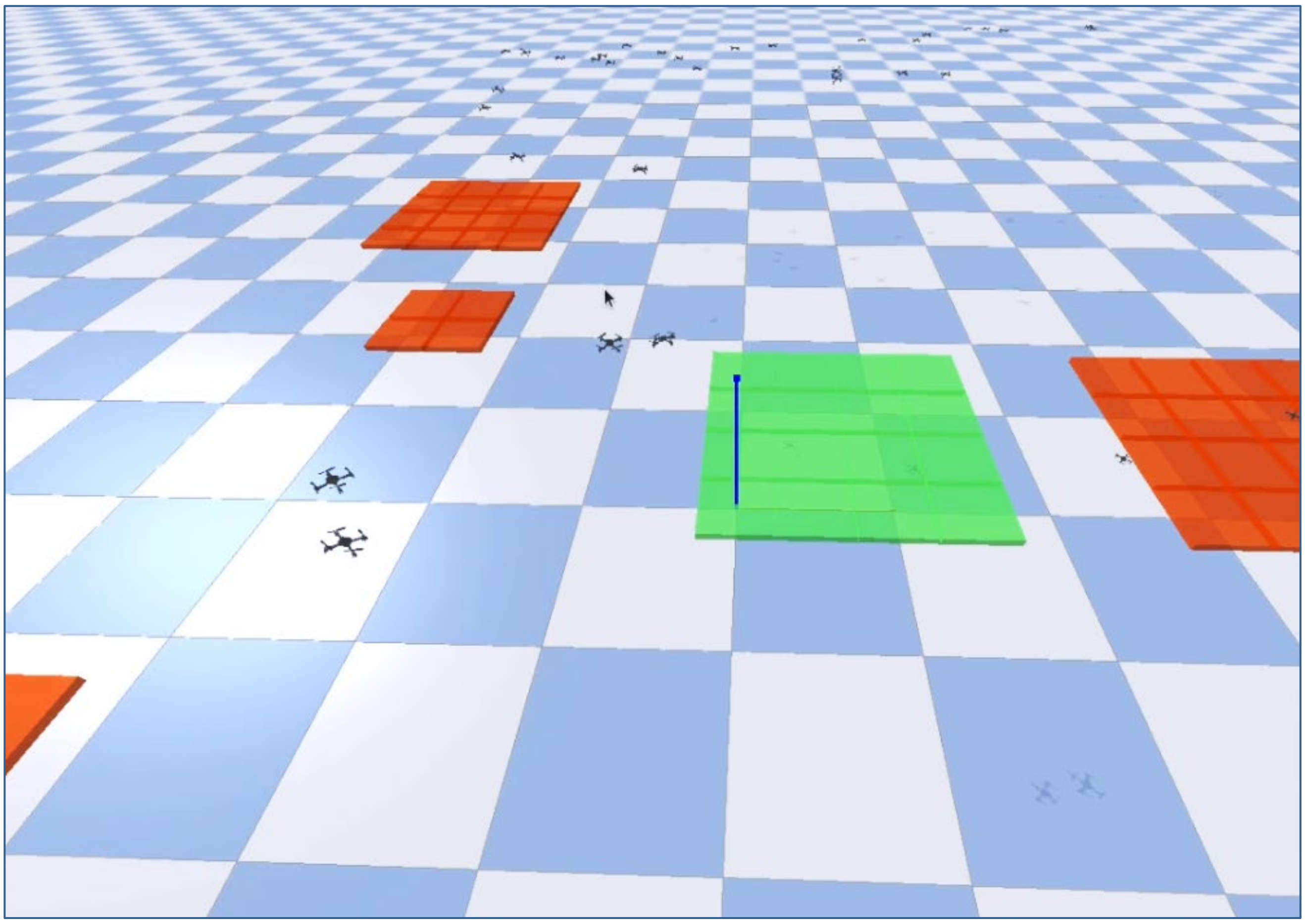


Figure 6: Rendering of a swarm of drones in PyBullet

Generation of Annotated Data for Off-Line Machine Learning Algorithms

Generation of annotated data for offline machine learning algorithms is another example that helps understand the benefits of parallelisation. In this case, the data acquisition stage is separated from the learning and validation stages and simulators are used to generate a large amount of synthetic data that can later be used. A particular technique of data acquisition called ***domain randomisation*** - where different aspects of the environment are randomised - was implemented to introduce variation into the learning data set and to force the machine learning model to handle many small visual variations, making it more robust (Figure 7). To generate sample synthetic data, we connected NRP Core and Unity Engine and used Unity Perception package.

Rendering of high-fidelity images may be time-consuming. Parallelisation of simulations via multiple Engines or multiple instances of NRP Core can definitely help address this problem by speeding up the data generation process. With this experiment, another relevant feature of NRP was tested, the DataTransfer Engine. This feature allows users to publish data (frames) onto an MQTT broker. The data can then be fetched by another Client, with the goal of storing it in a database, for instance.

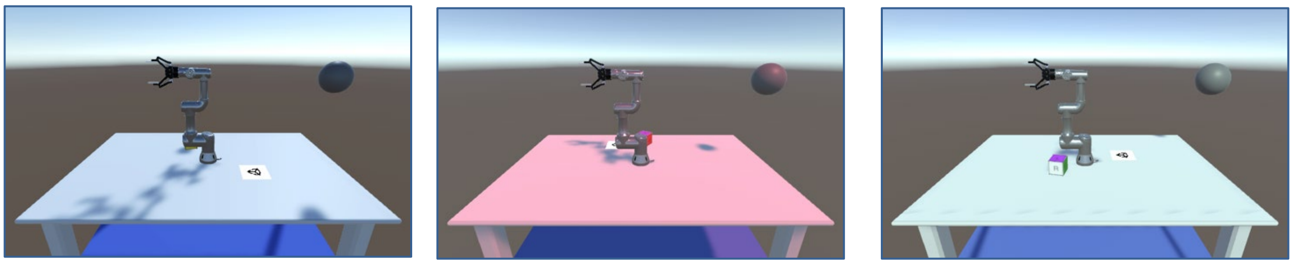


Figure 7: Examples of randomised synthetic data using Unity Perception package

The above experiments show how, thanks to the versatility and modularity of NRP v4.x, NRP users can now easily connect to powerful engines such as Unity, NEST, PyBullet and many others. Indeed, connecting heterogeneous simulation components in a hierarchical manner – for example, connecting low- and high-level controllers in the same simulation - is becoming increasingly common in both machine learning and robotics projects. Additionally, the new NRP Client allows for new workflows, e.g. centralised learning with multiple complex environments, where NRP Core with all its Engines becomes a service to the Main Script. Together with the ability to implement concurrency, and with the mechanisms that allow for distribution of different components, this will enable NRP users to train their model much faster and to fully exploit the power of their computing infrastructure.

3.2 Integration of neurocomputational models for reuse in the NRP

The new architecture of NRP v4.1 is especially suited for ***reuse***, ***extension*** and ***integration*** of computational models in neuroscience thanks to its capacity to implement highly modular simulations comprising both data-driven and model-based components.
In the context of Task T5.10, a number of closed-loop models provided by HBP Partners were integrated in the NRP and can now be reused as components (Engines), in new experiments.

Online Trajectory Generator for Robotic Control Pipeline

To support the work carried out by HBP Partners in Work Package 3 (WP3), we developed an Engine that integrates the ***mj_se3_tracker library*** created by HBP Partner SSSA (Scuola Superiore di Studi Universitari e di Perfezionamento Sant'Anna). This Engine provides NRP users with an online trajectory generator that can be used upstream to an inverse kinematic solver in a robotic control pipeline. The generated trajectories show a minimised jerk of the hand effector, thus offering the smoothness, human-likeness, and energy efficiency qualities that make them appropriate for a wide range of applications in robotics, especially when safe and natural movements are required. A second Engine was later provided, to monitor and analyse the output of the trajectory generator Engine (Figure 8).

Brain-Inspired Single Learning Trial Navigation Model

As described in Coppolino and Migliore, 2023[8], HBP Partners were able to successfully implement the integration of a brain-inspired Single Learning Trial (SLT) navigation model. In this experiment the authors have mimicked the neuronal architecture as well as connections of the hippocampus to control a robotic platform, simulated on the NRP, capable of learning while navigating a specific space (a four-arms maze) in the same way humans do. The simulated hippocampus model can alter its own synaptic connections while controlling and moving a car-like virtual robot which learns, in a

[8] Coppolino, S., and Migliore, M., (2023). *An explainable artificial intelligence approach to spatial navigation based on hippocampal circuitry*, Neural Networks https://doi.org/10.1016/j.neunet.2023.03.030. PLUSID: P3901

single trial, the correct path to reach an exit while navigating the space [9]. The simulated model needs to navigate to a specific destination only once before it is able to find the exit path (Figure 9). This is a remarkable improvement over current autonomous navigation systems that rely on deep learning and must calculate a multitude of possible paths instead.

For this integration the efforts required consisted of:

- porting of PyNN-based original neural network implementation to NEST 3 so to make it compatible with NRP's NEST Engine;
- fully re-implementing the driver experiment. The experiment code used by the original model had proven to be very brittle and troublesome for modifications only, given that the logic and many of its parameters were hard-coded. Therefore, we had to re-implement the experiment.

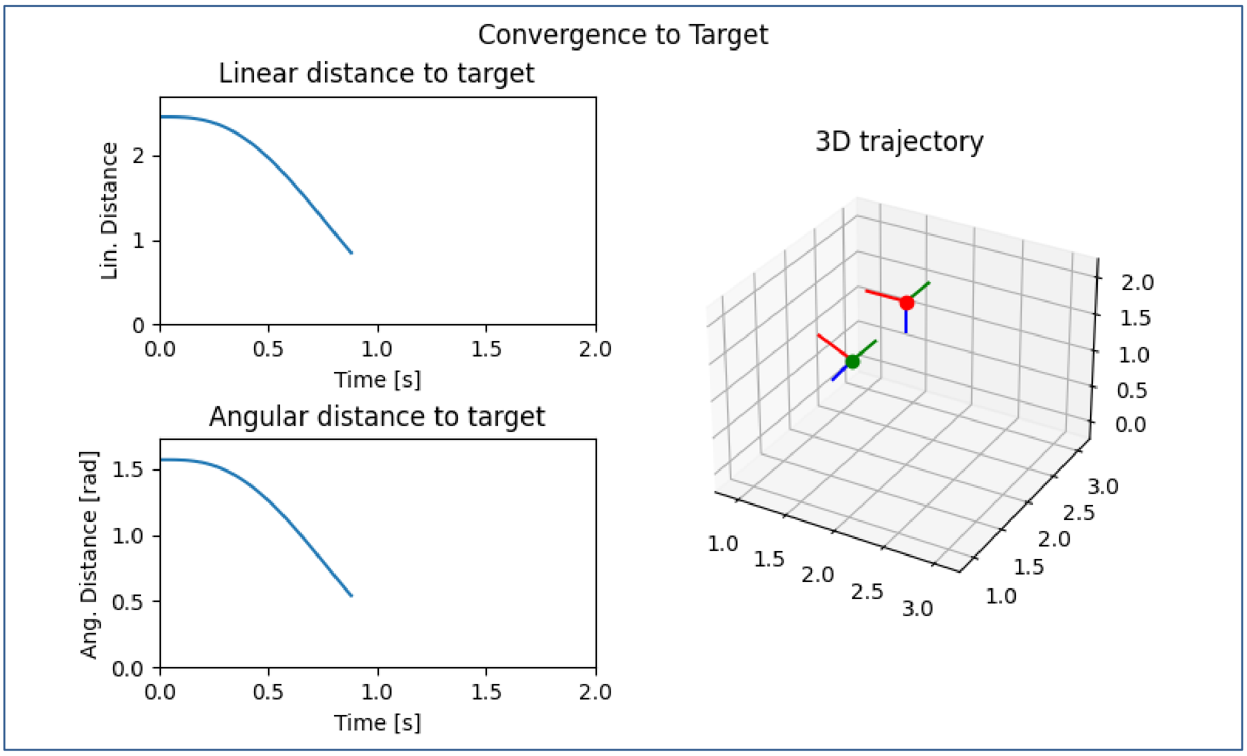


Figure 8: Minimum-jerk trajectory generator Engine plotter

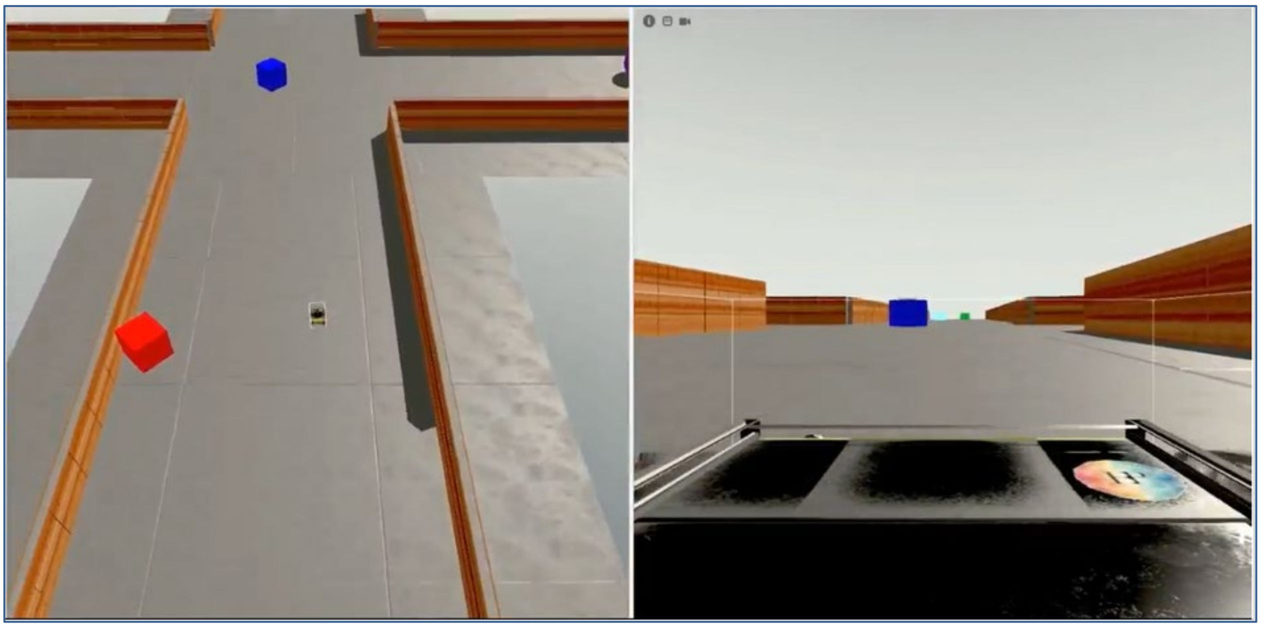


Figure 9: Brain-inspired navigation with NRP (Coppolino and Migliore, 2023)

[9] A video is available at https://www.youtube.com/watch?v=0Gl-HCrPLho

Visual Saliency Model

After evaluating several closed-loop models developed in Task T3.4 for their showcase experiments, the integration of a visual saliency model was finally considered for integration with the NRP. This model is now available to NRP users as an Engine that can be employed in their simulations.

A visual segmentation model also produced by partners in WP3 was briefly considered. Unfortunately, this model proved to be too specialised to be offered as a reusable component. In fact, it could segment only the objects used in scene of the showcase experiment for which it was developed.

The Visual Saliency model we integrated is a TensorFlow-based Engine. It processes images of a scene and outputs saliency maps (Figure 10). Saliency maps are of great importance for robotics applications where robots need to interact with and understand their visual environment. By focusing their attention on salient regions, robots can prioritise their perception, object recognition, and decision-making processes. This helps robots efficiently allocate computational resources while interacting with their environment.

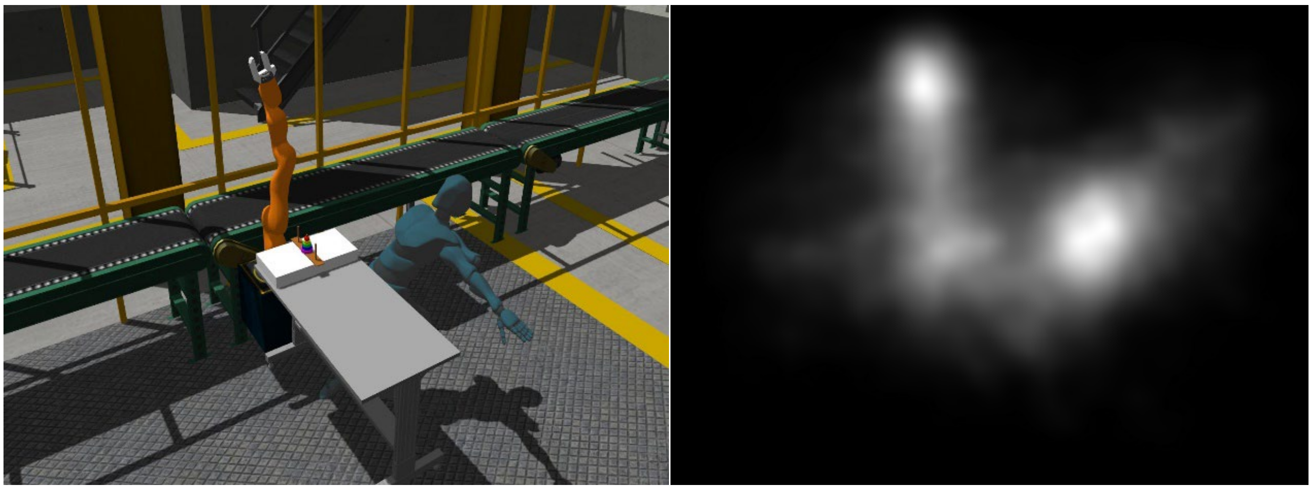


Figure 10: Visual Saliency Model, input (left image) and output saliency map (right image)

Spino-Cerebellar Model for Motor Learning and Control

The spino-cerebellar model (SC) developed in Bruel et al. (2023)[10] was reused to control a simulated musculoskeletal upper limb performing a set of motor tasks involving two degrees of freedom Figure 11. The movements generated with the integrated spino-cerebellar model had demonstrated improved motor control capabilities compared to those simulated using a cerebellar model only. Indeed, the authors showed that the use of the SC model facilitates faster convergence in motor learning and simpler cerebellar synaptic weight distributions, leading to effectively modulated muscle co-contraction, which enhance the robustness of the system against external motor perturbations.

To port this experiment to NRP v4.1, we developed an Engine connecting the NRP to the EDLUT simulator[11]. The spino-cebellar model consisted of two distinct engines: an OpenSim-based Engine for the spino-cerebellar component and the above mentioned EDLUT Engine for the cerebellum.

Additional efforts were required to reimplement the driver of the original experiment, due its original dependency to an external clock to synchronise its different components.

[10] Bruel, et al., (2023) The spinal cord facilitates cerebellar upper limb motor learning and control; inputs from neuromusculoskeletal simulation, bioRxiv https://doi.org/10.1101/2023.03.08.531839.
[11] The EDLUT (Event-Driven simulator based on Look-Up-Tables) is an advanced tool that allows the simulation of biologically plausible spiking cell models by using two different strategies: time-driven and event-driven based on look-up tables.

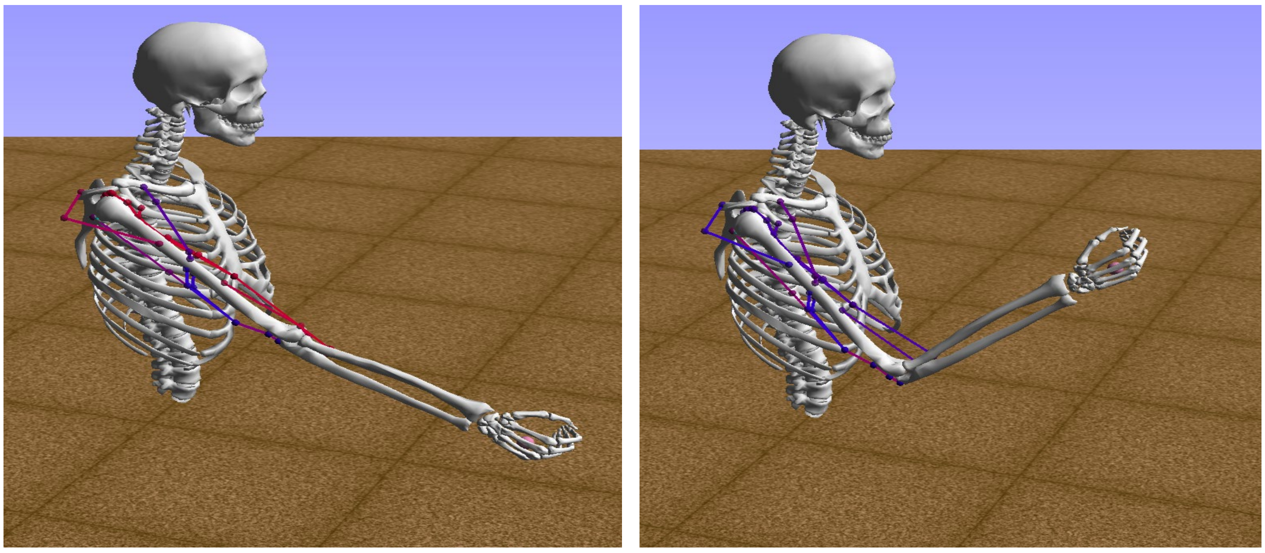


Figure 11: Spino-Cerebellar Model, at initial state (left) and at learning state (right)

3.3 New tools for education in computational neuroscience

The NRP potential as a learning and educational tool was significantly and further enhanced by dedicating specific efforts to integrate NRP v4.x with NEST Desktop technology.

NEST Desktop is an advanced web-based interface specifically tailored for **NEST Simulator**, which facilitates the construction, simulation, and analysis of neural networks without requiring extensive programming knowledge. It is designed for researchers, educators, and students who want to engage in neural simulations. NEST Desktop offers a user-friendly graphical user interface (GUI) for configuring the neural networks which ensures compatibility between the network configuration in NEST Desktop and the experiment settings in the NRP. Indeed, the GUI of NEST Desktop provides an intuitive environment for visualising and analysing the neural network architecture, thereby improving the users understanding.

The NRP provides a simulation framework, instrumental to the embodiment of the neural networks configured in NEST Desktop, it enriches NEST neural network model with dynamic capabilities and facilitates complex interactions between neural networks and the environment, thereby allowing the users to witness the implications of neural behaviour in a simulated realistic setting. This is essential for researchers and educators who wish to study neural network behaviour not in isolation but in response to various stimuli and in different scenarios.

Since both the NRP and NEST Desktop function as independent clients for NEST Simulator, and to simplify the process of installation and the communication setup, we created a ***Docker Compose configuration file***. This file enables users to quickly establish the necessary connections and dependencies between the two platforms.

An essential feature of NEST Desktop is the incorporation of the **Insite module**[12] for displaying spike activities. The Insite module is added as a layer to NEST Simulator for visualising neural network activity. When users upload the neural network configuration to NEST Simulator through NEST Desktop, a connection is established with the Insite module. At this point, the NRP utilises the pre-uploaded neural network in the NEST Simulator for the actual simulations. This integration allows for real-time visualisation of spike activities within the neural network, enhancing the user ability to monitor and understand the behaviour of the neural network during simulations.

[12] https://vrgrouprwth.github.io/insite/

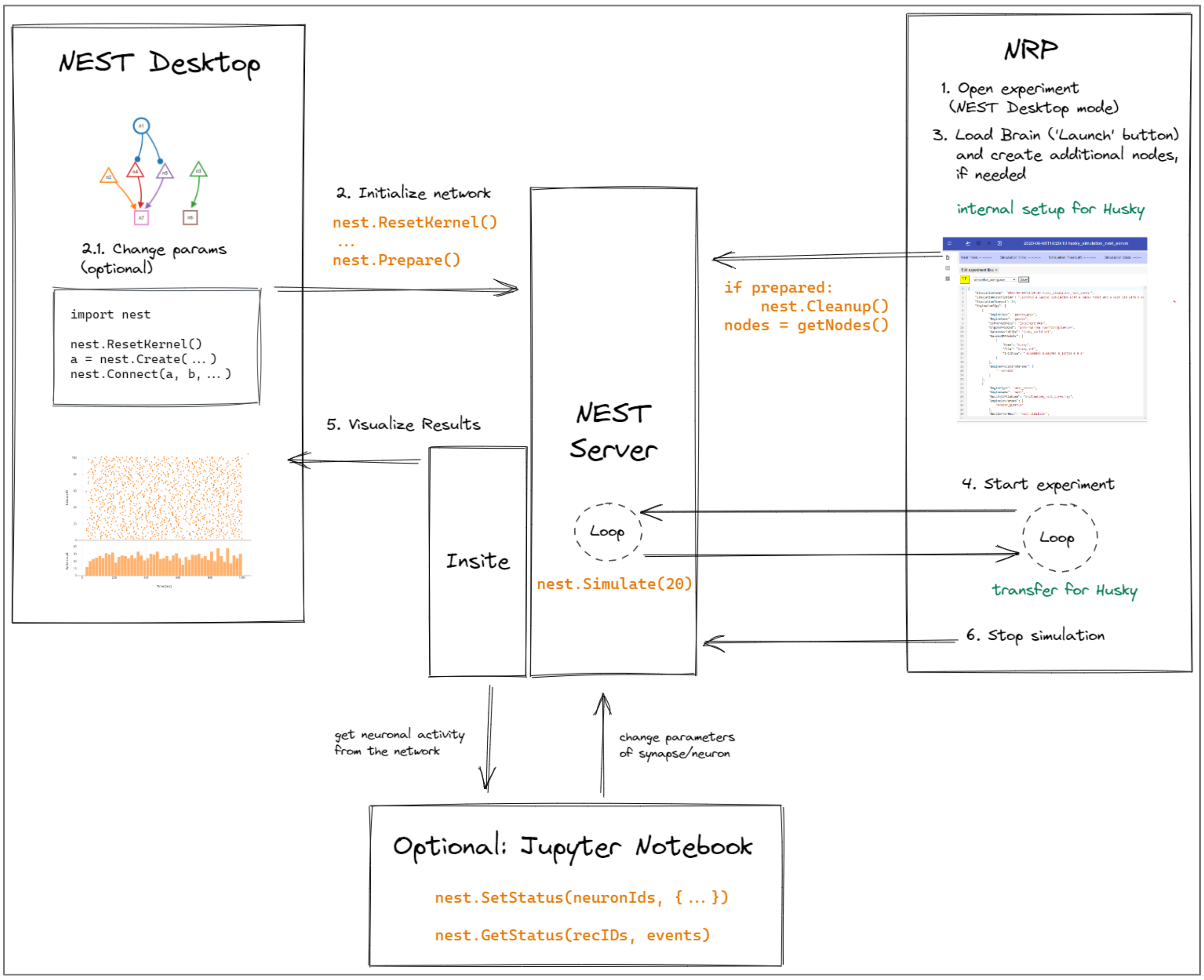


Figure 12: Workflow of the joint work of the NRP and NSET Desktop

The following steps summarise the NRP-NEST Desktop integration workflow (Figure 12):

1) Select and launch the desired experiment in the NRP - it is extremely important to select an experiment that is well-defined and compatible with the chosen neural network
2) Create or load the neural network in the NEST Desktop (it can be built from scratch with the NEST Desktop GUI or loaded from a file, but must be compatible with the selected experiment)
3) Upload the neural network from NEST Desktop to NEST Simulator
4) Run the simulation in the NRP
5) The NEST Desktop is now ready to receive the spikes activity visualisation data from NEST Simulator

Simulation in the NRP must be paused if neural network modifications are required. While the simulation is on hold, users can update the neural network in NEST Desktop using its intuitive GUI. However, it is crucial to ensure that the modified neural network remains compatible with the experiment settings in the NRP. This compatibility guarantees seamless integration between the modified network and the NRP simulation environment.

Once the modifications are made, the updated neural network is uploaded to NEST Simulator via NEST Desktop. The experiment can then be resumed in the NRP, utilising the revised neural network that adheres to the experiment configuration.

Jupyter Notebook can also be connected to NEST Simulator and the Insite module as another client. This provides users, especially researchers and educators, with an additional platform to interact with neural networks and visualise their activity. With Jupyter Notebook users can write scripts and codes to further customise and analyse the simulations in a more programmatically controlled environment, making it a versatile tool for advanced neural network simulations and analyses.

4. Looking Forward

The NRP has been a great learning experience inside the HBP for all those who contributed to its development. Many lessons were learnt in terms of how to communicate with neuroscientists, what their expectations were (not only in terms of software features, but also in terms of turnaround time), and what their investment threshold was in terms of the effort required to become a proficient user of a new software platform. As such, we noted that, even when *a priori* attracted by the possibility of embodiment, many were reluctant to bear with the growing pains of the NRP. In particular, it became clear that the devil is in the details insofar as every experiment requires some unique implementation features, and that as such the NRP needed to accommodate these idiosyncrasies in a more organic way than foreseen in the legacy version.

The result with NRP v4.1 is a framework that is truly hitting a sweet spot between modularity, distribution and usability. We believe that the unique features of the latest NRP version make it a particularly suitable tool to build upon further in neuroscience than what was done so far in the HBP. In particular, the NRP v4.1 uniquely enables the following:

- Prototyping of concrete applications of neuromorphic computing: embodiment in simulation is a powerful tool to evaluate the functionality of neuromorphic circuits in a safe manner before proceeding with validation on hardware – especially when human interaction is involved.
- Large-scale embodied simulations: the synergy between the chosen software architecture for NRP v4.1 and containerisation technology make it easier than ever to set up complex embodied simulations with large-scale brain models (in the order of millions of neurons and billions of synapses) on supercomputer[13].
- Reuse of brain and neuronal circuit models: a significant amount of effort is usually required to reuse an existing brain model. This is often due to incompatibility between software versions, dependency conflicts, inconsistent datatypes, or poor documentation. This state of affairs prevents not only reproducibility studies, but also reuse of said model beyond the scope of the experiment for which it was created. The NRP provides the generic scheme and interfaces to remedy this situation (e.g. through containerisation and subsequent container orchestration) and therefore support truly integrative neuroscientific endeavours in a straightforward, well-documented manner.
- Large-scale cognitive architectures: in the same spirit as above, the decoupling of implementation details from the functional purpose of the various components of a large-scale cognitive architecture facilitates investing its properties, transferring it to different tasks from the one it was initially conceived, and therefore focus entirely on studying the behavioural traits it underpins, rather than the minutiae of its engineering. The NRP thus appears to be the ideal framework for such scientific endeavours.

Additionally, ***the unique features of the NRP as an integrative framework also make it a particularly suitable tool for Digital Twins***[14].

A digital twin can be defined as the *digital representation* of an actual physical system through a combination of mathematical models and data. This digital representation supports understanding and prediction of the behaviour of its physical counterpart by testing and improving the design and functionality of such system without the need for physical prototypes – or, when prototypes cannot be used -, and without the actual system incurring any risk, therefore with extremely reduced costs and development time. The essential distinction between a digital model and a digital twin is the

[13] Feldotto, et al., (2022) *Deploying and Optimizing Embodied Simulations of Large-Scale Spiking Neural Networks on HPC Infrastructure*. Front. Neuroinform. 16:884180. doi: 10.3389/fninf.2022.884180 PLUSID: P3266

[14] Amunts, et al., (2023). The coming decade of digital brain research - A vision for neuroscience at the intersection of technology and computing (Version 4.0). Zenodo. https://doi.org/10.5281/zenodo.7764003 PLUSID: P3321

bidirectional data flow (*digital bridge*) between the physical system and its digital representation which conveys *real-world feedback*, i.e. information not only about the physical system itself, but of what happens to the physical system in the real world. This bidirectional communication flow must be a mechanism (e.g. sensors) that reliably reflects any modification made to the physical system onto its digital twin and, conversely, for simulation of the latter, must provide actionable information to the physical system. The introduction of digital twins unlocks unprecedented possibilities for simulation and AI. For example, in manufacturing, a digital twin may inform the whole product lifecycle management, or the implementation of smart human-robot collaboration mechanisms. In the automotive industry, the digital twin of a car engine can underpin new processes of predictive maintenance. In healthcare, digital twins might help predict the pharmacological effects of drugs, or better understand disease mechanisms. Thanks to its improved and new features – real-time communications, parallel and containerised distribution of simulation instances, client-server communications -, NRP v4.1 can now become the go-to open access tool for generation and simulation of digital twins in such a wide variety of cases.

Another field in which we believe NRP v4.1 will prove to be a valuable asset to innovators is ***in silico design, testing and certification of AI-enabled medical devices***.

On January 2023 the EU-funded project TEF-Health, Testing and Experimentation Facility (TEF) for the integration, testing and validation of advanced AI-based technologies and robotics technologies for health and care[15], was launched. With a budget of EUR 60 million, TEF aims to radically improve effectiveness, resilience and sustainability of European health and care systems by speeding up the development and the market access of trustworthy AI solutions in Europe. As partner of TEF, TUM (Technische Universität München) is using NRP v4.1 as one of the tools available to the consortium for development and testing of facilities and AI based services in healthcare.

We hope and trust that these exciting possibilities will be explored in the near future by an ever-growing NRP user community, whom we thank for having accompanied us on this exciting journey that was the HBP.

15 https://www.brainsimulation.org/bsw/zwei/news/single/11263-tef-health-project-awarded-60-million-for-testing-ai-and-robotics-technologies-in-healthcare