A guide to VCBot

Hendry F. Chame, Ahmadreza Ahmadi, Jun Tani

Okinawa Institute of Science and Technology Graduate University (OIST)
Cognitive Neurorobotics Research Unit (CNRU)
1919-1, Tancha, Onna, Kunigami District, Okinawa 904-0495
E-mail: {hendryfchame,ar.ahmadi62,tani1216jp}@gmail.com
April 2020

Introduction

The *virtual Cartesian robot* (VCBot) is a program designed for interacting with an artificial neural agent through the computer mouse, serving diverse purposes such as entertainment, educational, and research, among others. As shown in Fig. 1, VCBot is inspired by the structure of a 3D Cartesian robot actuated by prismatic joints [1].

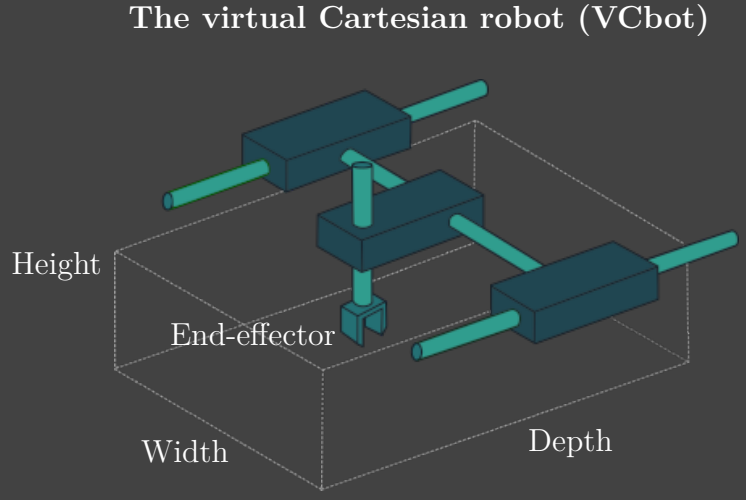

Figure 1: Representation of a 3D Cartesian robot motorized by prismatic joints. In VCBot only two degrees of freedom are controllable in the interaction (the width and depth dimensions), so the height dimension is kept constant.

Several features make VCBot an interesting tool. Training datasets can be constituted by kinesthetic demonstration of the desired primitives, that is, by virtually moving the end-effector to shape the behavior patterns. Thereby, the user is able to participate in the design of tasks, which may enrich the interaction experience. Experiments are recorded in real-time, including data from emergent behavior, the robot desired actions and internal states, and the human desired actions. Thus, data can be lately analyzed, which is relevant to research

purposes. The sections below describe the software architecture, the methodology proposed, and provide some hints of how to get the most out of VCBot.

Software architecture

VCBot is implemented in the Python programming language (see Fig. 2), running on the top of the *neural robotics library* (NRL), an open-source tool implemented in the C++ programming language for prototyping human-robot interaction experiments, based on artificial neural cognitive control, from recurrent neural network models. The reader interested in consulting the theoretical foundations of NRL is referred to the article "Towards hybrid primary intersubjectivity: a neural robotics library for human science" [2]. Technical details on how to proceed with the installation of the software in provided in the *readme* file of the VCBot[1] and the NRL[2] repository projects.

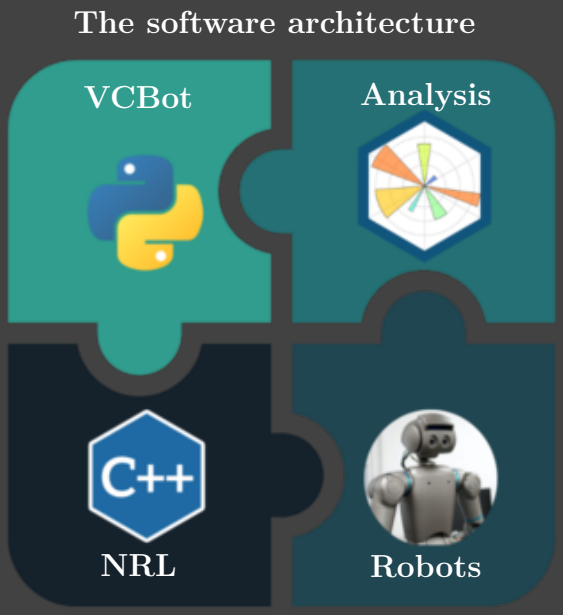

Figure 2: VCBot runs over NRL which performs the core computations for cognitive control. NRL is implemented in the C++ programming language to ensure efficient use of hardware resources for real-time experiments with robots. VCBot is developed in the Python programming language version 3, given the multi-platform portability of the graphical user interface and the availability of analytical resources.

Graphical user interface (GUI)

The VCBot project is structured around the methodology illustrated in Fig. 3, which involves four steps: *Modeling*, *Training*, *Experiment*, and *Analysis*. Thus, the GUI was designed

[1]VCBot repository: https://github.com/oist-cnru/VCBot
[2]NRL repository: https://github.com/oist-cnru/NRL

inspired by a notebook layout metaphor, where each tab contains the controls relevant to a given methodology step. These functionalities are described below.

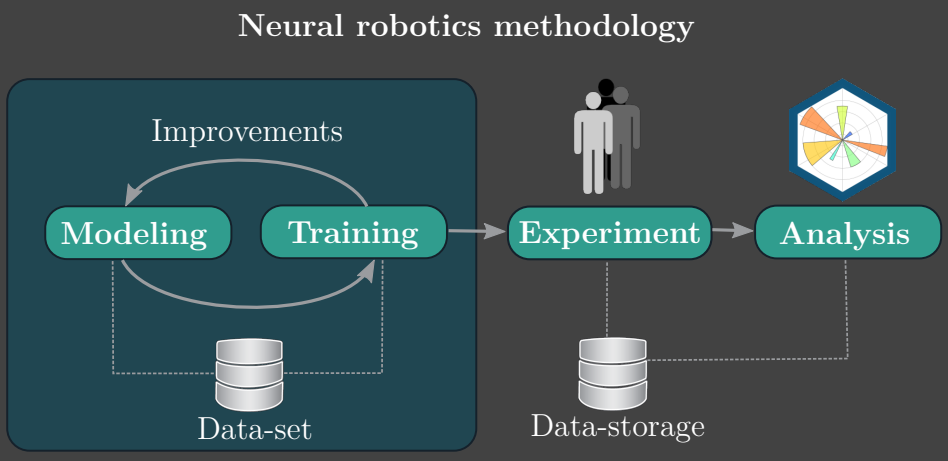

Figure 3: The user develops a prototype of the neural agent by selecting the model's meta-parameters, capturing the behavior dataset, and training the model. In the experimental phase data from real-time interaction is saved, and lately analyzed with the help of statistical and graphical tools.

Modeling

There are several sorts of artificial neural networks available. NRL implements by default a type *predictive-coding-inspired variational recurrent neural network* (PV-RNN) [3]. Constituting PV-RNN model requires the user to select the architecture parameters, in order to profile important qualities such as the *cognitive compliance* of the agent [4], and the learning of hidden probabilistic structure in data. Unfortunately, there is no analytical method for selecting the parameters, so it should be done by trial-and-error. In the *Modeling* tab (see Fig. 4) it is possible to set the parameters for multiple layers separated by comma, some guidelines are provided in Table 1.

A total of six operations can be performed in the *Modeling* tab: addition, clearing, browsing, selecting, detailing, and removing. Through the *Add* button a new model is created in the database. Text from the addition form can be erased by pressing the button *Clear*, no change is done to the database. The combo-box *Name* is a control widget for browsing models and visualizing their parameters. It is recommended to check the details of pre-included models in VCBot as a reference for the selection of parameters.

In order to activate the functionalities of some tabs (e.g. *Training*, *Experiment* and *Analysis*) it is required to select a model for operation by pressing the button *Select*. A model has to be associated with a dataset (which is described in the next section). For already trained models it is not required to reestablish the association (VCBot recognizes it automatically), so the user can proceed to operate other tabs of the notebook. When available, previous training can be visualized by pressing the button *Details*. Finally, the button *Remove* deletes a model from the database. This operation cannot be undone, so confirmation is required from the user.

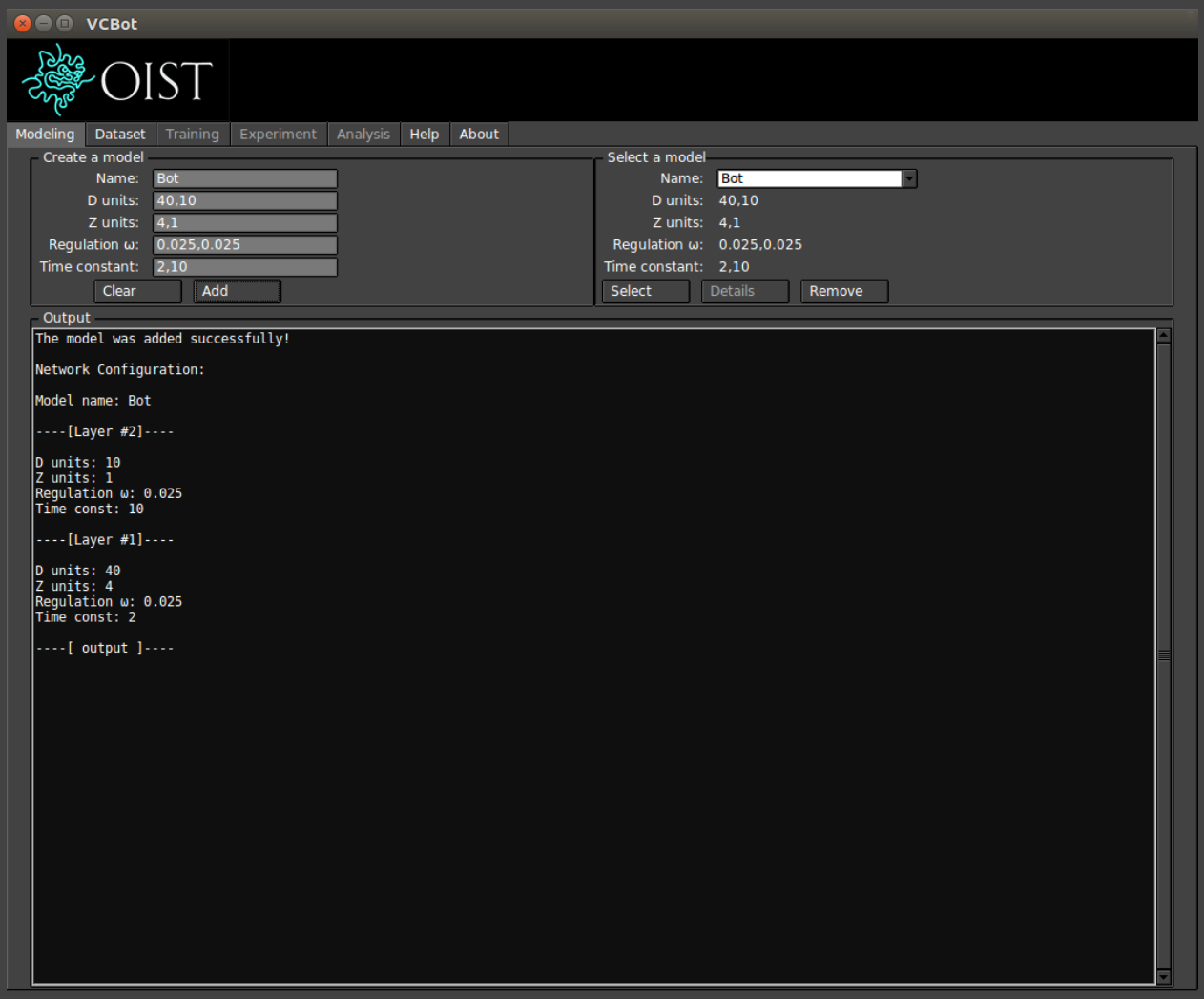

Figure 4: The user has performed an addition operation. The output widget provides feedback on the parameters selected for the model.

Dataset

The objective of the *Dataset* tab is to allow the constitution and management of training sets, containing behavior primitives (see Fig. 5). Five main functionalities are provided: addition, clearing, browsing, selection, and edition. For adding a new dataset, it is required to provide an alphanumeric name, to define the number of steps for the primitives (*Primitive length*), and to select the desired sampling period in milliseconds (*Period in ms*). The form text entries can be erased by pressing the button *Clear*. A new dataset is registered in the database by pressing the button *Add*.

In the *Selection* panel datasets can be browsed through the combo-box widget *Name*, so the number of primitives and details regarding captures is shown. Three actions are possible:

Table 1: **PV-RNN meta-parameters**

Parameter	**Description**
d units	Represent the deterministic latent state. As a rule of thumb they are set ten times more numerous than z units.
z units	Represent the stochastic latent state in the prior and posterior distributions. In PV-RNN these units encode a Gaussian distribution parameterized by a mean (or expectation) $\boldsymbol{\mu}$ and a standard deviation $\boldsymbol{\sigma}$, so $\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} * \boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon}$ sampled from $\mathcal{N}(0, 1)$.
Regulation w	Is a meta-parameter which influences the learning of the posterior and the prior distributions. In general, the higher the parameter is set, the more similar the hidden prior and posterior distributions would be, so the internal representation would be less sensitive to stochasticity during interaction (it would be less cognitive compliant to the partner's intentions). On the other hand, if w is set too low, the agent's generative process based on the prior distributions would be poor, so the agent would tend to behave erratically.
Time constant	The timescale conditions the temporal dynamics of the layers. The constants should be selected increasing proportionally between adjacent layers from the lowest to the highest, so low layers present faster dynamics than higher layers. For example, assuming a configuration of three layers, in case $\iota^{\text{middle}} = 5\iota^{\text{low}}$, then $\iota^{\text{high}} = 5\iota^{\text{middle}}$.

selecting, editing, and removing. By pressing the button *Select*, a dataset is associated with a model. When creating new associations it is required selecting first the model then proceeding to select the dataset. By pressing *Remove* the dataset is deleted from the database. The operation cannot be undone, so confirmation is required from the user. In case models were trained with a dataset removed afterwards, the association is automatically deactivated. Lastly, by pressing the button *Edit*, the *edition* section is enabled.

Three main tasks can be accomplished in the *Edition* panel: browsing, adding, and removing primitives. The combo-box widget *Primitive* allows the user to visualize primitives in the workspace from their identification key (an integer number). The addition of a new primitive starts by pressing the button *Add*, so VCBot begins to monitor the end-effector's position, according to the chosen sampling period. A green text box in the workspace's upper right corner pops up instructing the user to "Press the control key to start" recording. In order to register the demonstration the key *control* should be pressed and held while moving the mouse within the workspace boundaries. Once the user starts to record, a red text is displayed indicating how many time steps are still required to complete the primitive capture. The key *control* may be released anytime after the time counter reaches the desired number of steps. The trajectory trail changes from red to green if the sampling steps are completed (which indicates the primitive has been added to the dataset), otherwise, the recording is discarded. A primitive can be deleted by pressing the button *Remove*. This operation cannot be undone.

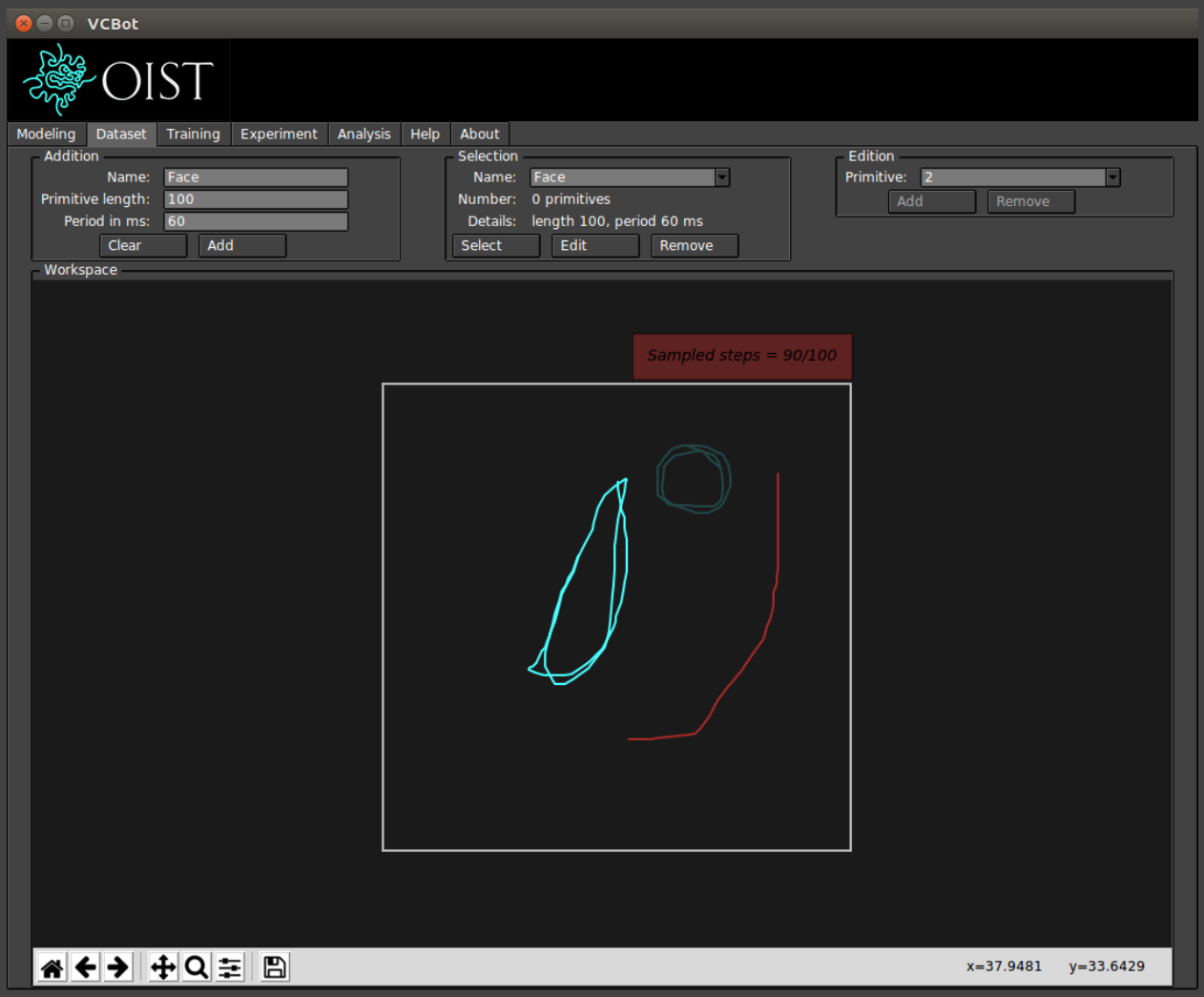

Figure 5: Visualization of a primitive recording for dataset constitution. In the workspace three primitives are shown. The red circle represented darker at the top left was the recorded earliest. The last visualized primitive is shown in light blue. In red, the current Recording.

Training

The *Training* tab allows the user to train a selected model. There are two panels available in the interface. The *Selection* panel provides feedback on the model-dataset association established. When previous training is available, data can be visualized by pressing the button *Previous training*. Following the Adam method for stochastic optimization [5], the first row in the panel *Parameters* allows the selection of the learning rate α. Since the algorithm updates the exponential moving averages of the gradient (a first moment estimation) and the squared gradient (the uncentered variance estimation), the hyper parameters β_1 and β_2 control, respectively, the exponential decay of these moving averages.

The *Training* tab interface

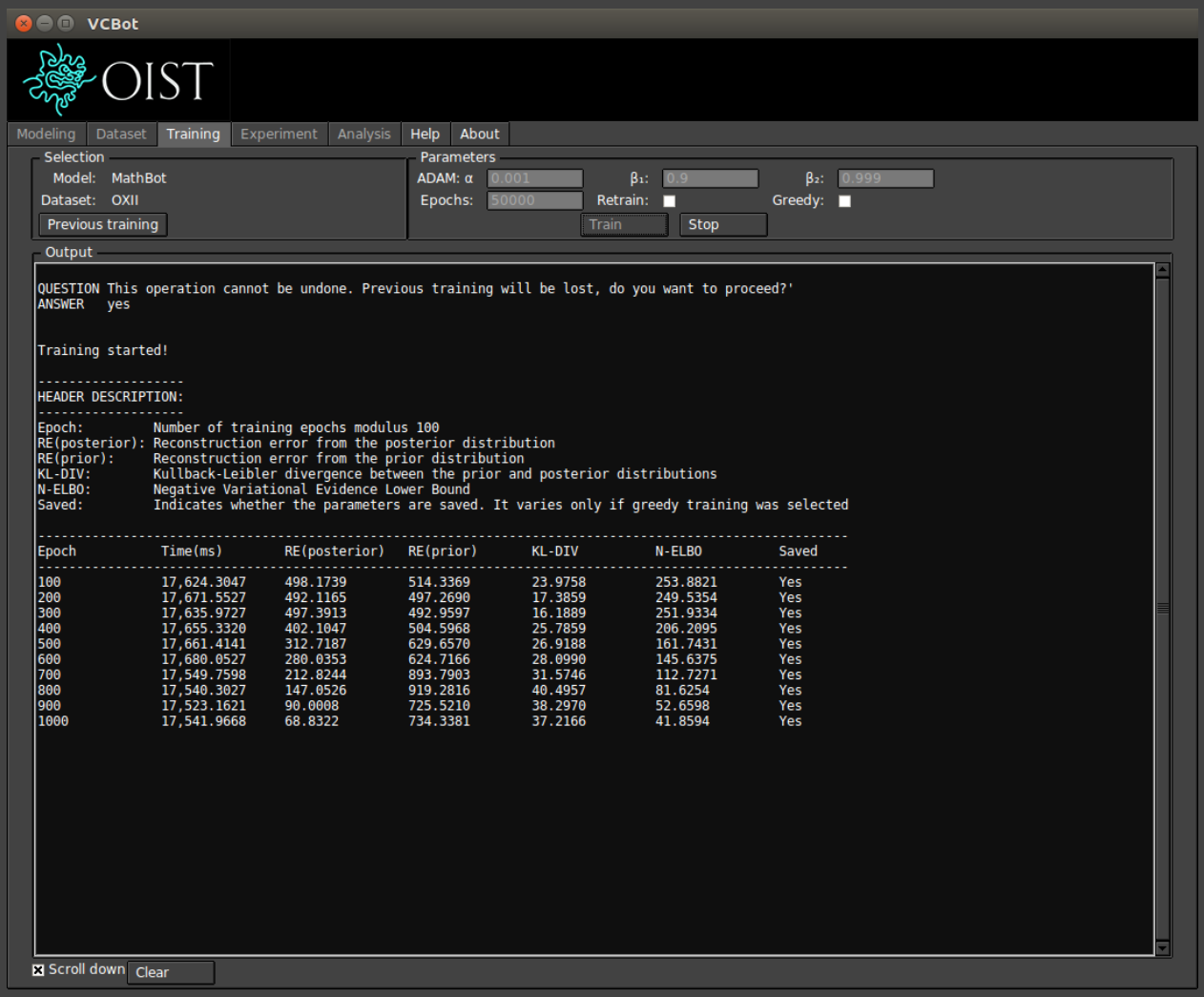

Figure 6: Visualization of the output computed for the first thousand epochs by pressing the button *Train*.

In the second row of the panel *Selection*, the desired number of training epochs can be selected in the text-box *Epochs*. Default parameter values are provided for reference in the text-boxes. The check-box *Retrain* allows the user to decide whether training from scratch or continuing from previous training, in the later case the number of epochs should be set higher than previous selections. The check-box *Greedy* specifies whether training is saved at each epoch modulus 100 (the training period at which data is saved in permanent storage for input/output efficiency purposes), or only if the actual loss function evaluation is optimal. Training starts by pressing the button *Train*, and can be canceled by pressing the button *Stop*. The evolution of the training process is shown in the output widget (see Fig. 6).

Experiment

The objective of the *Experiment* tab is to provide functionalities for performing and recording interactions. As shown in Fig. 7, four panels are available on the top section. The panel *Selection* indicates the current model-dataset association. Through the combo-box *Signal* the user selects the desired information for on-line visualization, which is described in Table 2. Since latent states may be multi-dimensional, in the combo-box widget *Format* it is possible to choose among three dimension reduction methods: *Raw* (selects data from the first dimension), *Sum* (summation), *Mean* (or expected value). The signal selection and reduction method can be changed on-line.

Table 2: **Information available for on-line visualization**

Signal	**Space**	**Description**
N-ELBO	$\mathbb{R}$	Negative evidence lower bound.
Reconstruction	$\mathbb{R}$	Reconstruction error from the posterior distributions.
Regulation	$\mathbb{R}$	Kullback-Leibler divergence between the prior and posterior distributions.
$\boldsymbol{d}^k$ (prior/posterior)	$\mathbb{R}^{\lvert\boldsymbol{d}^k\rvert}$	Deterministic latent state, with $\lvert\boldsymbol{d}^k\rvert$ the number d units selected for layer k.
$\boldsymbol{h}^k$ (prior/posterior)	$\mathbb{R}^{\lvert\boldsymbol{d}^k\rvert}$	Leaky integrator latent state.
$\boldsymbol{z}^k$ (prior/posterior)	$\mathbb{R}^{\lvert\boldsymbol{z}^k\rvert}$	Stochastic latent state.
$\boldsymbol{\mu}^k$ (prior/posterior)	$\mathbb{R}^{\lvert\boldsymbol{z}^k\rvert}$	Mean of the hidden Gaussian distribution.
$\boldsymbol{\sigma}^k$ (prior/posterior)	$\mathbb{R}^{\lvert\boldsymbol{z}^k\rvert}$	Standard deviation of the hidden Gaussian distribution.
$\log \boldsymbol{\sigma}^k$ (prior/posterior)	$\mathbb{R}^{\lvert\boldsymbol{z}^k\rvert}$	Logarithm of $\boldsymbol{\sigma}^k$.
$\boldsymbol{\epsilon}^k$ (prior/posterior)	$\mathbb{R}^{\lvert\boldsymbol{z}^k\rvert}$	Noise sampled from $\mathcal{N}(0, 1)$.

In the panel *Adam* the values for the Adam optimization algorithm [5] can be set. As already explained in the *Training* tab section, the algorithm updates the exponential moving averages of the gradient (a first moment estimation) and the squared gradient (the uncentered variance estimation), so the hyper parameters β_1 and β_2 allow to control, respectively, the exponential decay of these moving averages. It is important noticing that experiments usually require distinct values for the parameters, which is explained next.

The panel *Postdiction* allows the user to activate or deactivate the capacity of inference in the robot, through the combo-box *Enable*. In case postiction is disabled, the robot generates behavior exclusively from the prior hidden distributions, so it would be insensitive to the human intended actions. Although this is not interesting for interaction, disabling postdiction can be useful to verify the quality of behavior generation learned during training. Once postdiction is enabled, the number of free energy optimization epochs and the size of the temporal sliding window can be selected in the text-fields *Epochs* and *Window*, respectively. As detailed in Table 1, the parameter w regulates learning the posterior and the prior distributions. However, as described in [2], learning takes place with distinct purposes during the training and experiment phases (see Fig. 3). Learning in the training phase in-

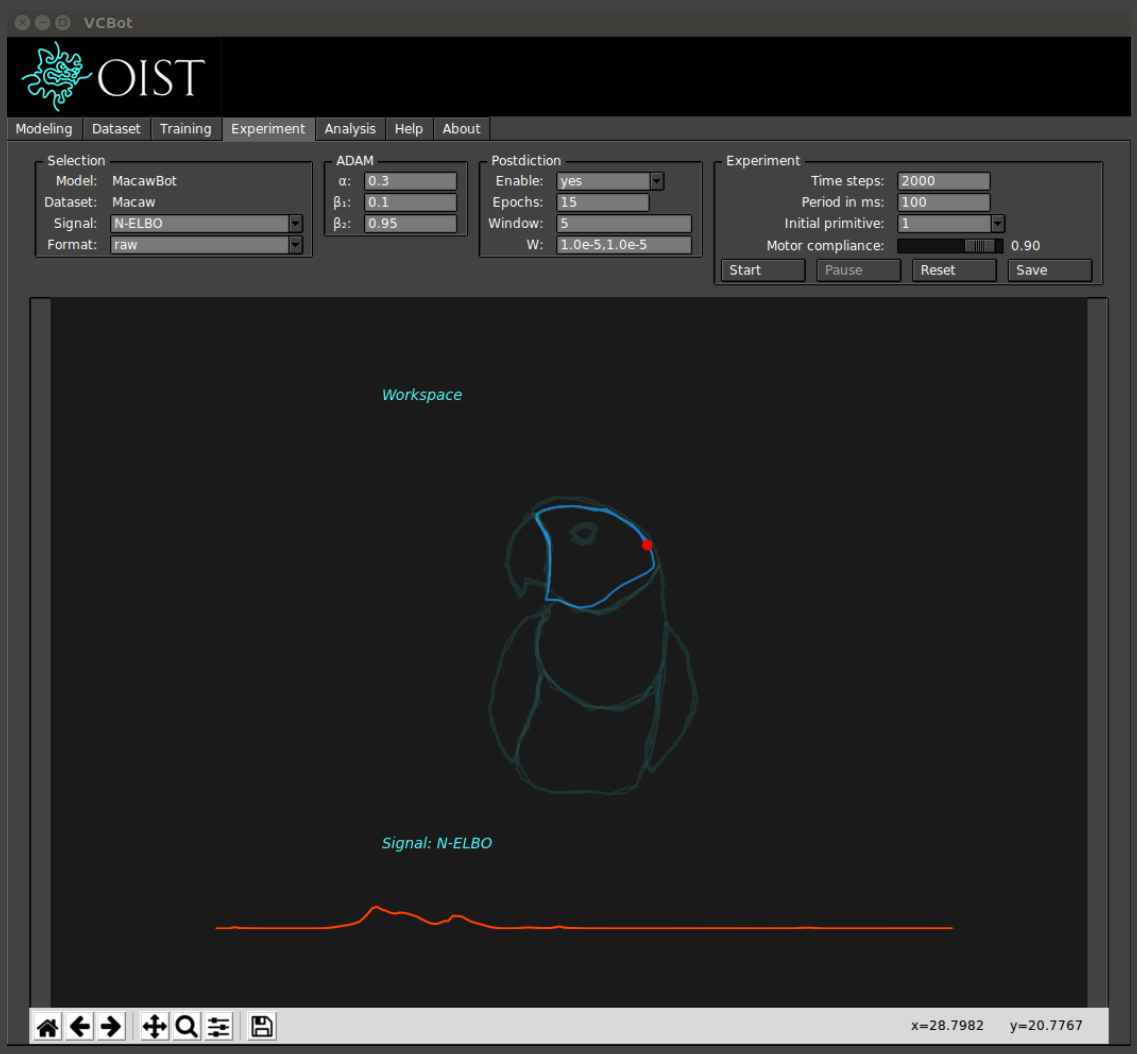

Figure 7: Screen capture of a real-time interaction. Two plots are rendered in the central canvas: the workspace on the top and the on-line time evolution of free energy optimization or latent states data on the bottom (as selected by the user). The workspace shows the training dataset in darker color for reference, which includes behavior primitives globally shaping a macaw cub. The end-effector is represented by the red circle. Recent behavior is shown in light blue (the robot is reproducing the shape of the bird's head). The signal shown in the lower plot corresponds to the negative evidence lower bound N-ELBO.

volves the off-line modification of parameters, including synaptic weights and state variables. Since in the experiment phase on-line inference is computed, only state variables within the temporal sliding window are learned, so changes are induced in the latent representations. Consequently, the values for the parameters w (as well as in the panel *Adam*) can be set differently to achieve the experimental purposes in the methodology.

In the panel *Experiment* the user can select the number of time steps to be recorded

in the experiment (the field *Time steps*), and the desired sampling period in milliseconds (the field *Period in ms*). It is recommended to set the sampling parameter the same as for capturing the behavior primitives in the dataset. Through the field *Initial primitive*, the user can influence the initial behavior of the robot by inducing the generation of a particular primitive. It is not guaranteed that the robot will proceed accordingly, as it does when postdiction is disabled. This is due to the fact that on-line inference is subject to sensory stochasticity, so the internal latent states of the PV-RNN model could fall in a distinct limit cycle attractor region.

The position of the end-effector emerging from the interaction is determined as the linear interpolation of the human and the robot desired actions. By selecting the factor γ in the sliding bar *Motor compliance*, the user can influence emergent behavior, such that

$$position = \gamma human - (1 - \gamma) robot. \tag{1}$$

Otherwise expressed, the closest the selection of γ is to one, the more emergent behavior is determined from the human's intended actions, contrarily, the closest the selection is to zero, the more control would be exerted by the robot's intended actions. In order to avoid brusque changes in the end-effector, in VCBot the rate of change in position is saturated by a constant factor.

An experiment can be started by pressing the button *Start*. It is recommended to press the button *Reset* previous to begin a new capture. In doing so, the program clears cached data and resets the time step counter. The experiment can be paused by pressing the button *Pause*, and resumed by pressing the button *Start*. Data is automatically saved in persistent storage once the time counter reaches the desired time steps. In case the user is satisfied with the actual capture (before ending the experiment), it is possible to launch the data saving work-flow manually by pressing the button *Save*. Likewise with dataset recording, interaction is enabled by pressing the key *control* and moving the mouse. For this, a green text box message in the upper right corner of the workspace pops up instructing the user to "Press control to interact".

Analysis

The *Analysis* tab provides graphical and statistical tools to visualize experimental data. The *Selection* panel indicates the model-dataset association established (see Fig. 8). Through the combo-box widget *Experiment* the user can browse captures, which are named after the date and time stamps of recording. Since models can have multiple layers, the combo-box widget *Layer* allows to choose between showing all layers (in the option *All*), and a particular layer numbered increasingly from the bottom-most to the top-most. By pressing the button *Select* a dialog window pops up allowing the user to choose which latent states from the layer are going to be plotted (see Fig. 9). As detailed in Table 2, the latent states can be multi-dimensional. Hence, in the combo-box widget *Format* it is possible to choose among three dimension reduction methods: *Raw* (selects data from the first dimension), *Sum* (summation), *Mean* (or expected value). Lastly, the plot corresponding to the time evolution of the signals is rendered by pressing the button *Plot*.

Basic statistical estimations can be computed for all the latent states and data captured

The *Analysis* tab interface

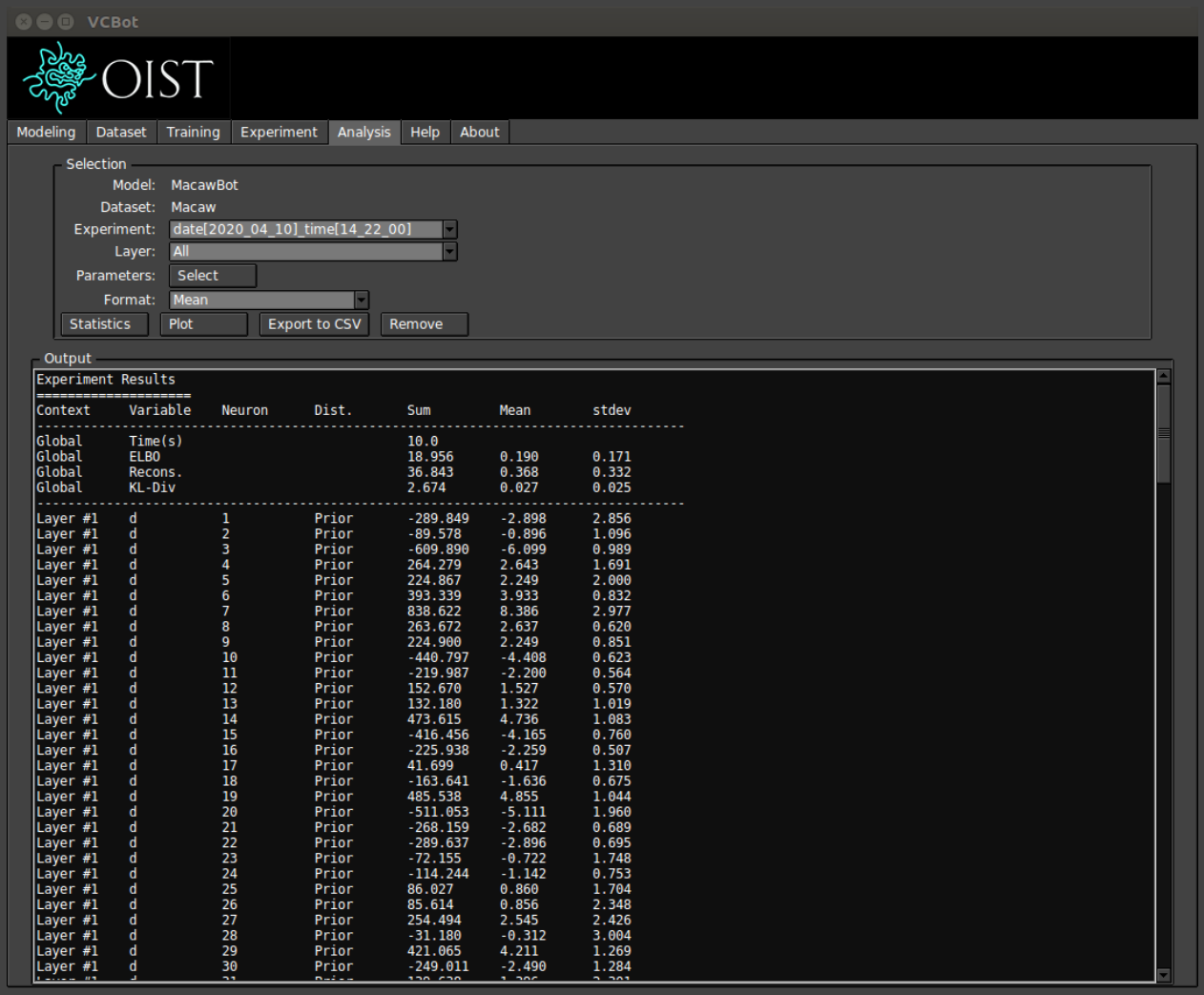

Figure 8: Visualization of the output computed by pressing the button *Statistics*.

from the optimization of free energy, by pressing the button *Statistics*. As illustrated in Fig. 8, the summation of multi-dimensional data, the mean, and the standard deviation are provided in the output widget. In VCBot, experimental data is recorded in Python's *numpy* package file format (*npy* extension), which is non-human readable. However, as detailed in the section named "The file-system explained", by pressing the button *Export to CSV*, the user can save data into the comma-separated value (CSV) format, which is human readable, and can be imported in conventional statistics software platforms for further analysis.

Exit the application

There are two ways of closing the application. The first one is through the mouse interface, by pressing the close icon in the application's window top bar. It is also possible to finish

The layer state selection dialog

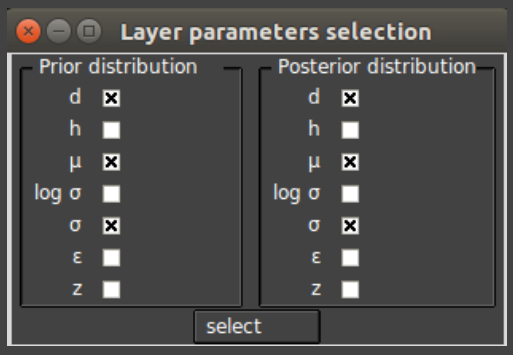

Figure 9: The user can select the desired states (see Table 2) by clicking on the check-boxes, and registering the selection by pressing the *Select* button.

the program execution by pressing the *Esc* key. In either way a Yes/No dialog requires the user to confirm the operation.

The file-system explained

At earliest design stages, the VCBot project was conceived as an open-source research tool for demonstrating the functionalities provided by NRL. Once the project started to evolve, and the potentialities of the Python platform became more evident to us, it remained the feeling that VCBot was becoming interesting by its own qualities. Hence, in order facilitate adaptations to other project's requirements, in this section the client file-system is reviewed. The information provided is an *under-the-hood* sort of description which is not required to operate the GUI. The reader interested in knowing more details about the implementation is invited to consult the program sources, and to check for additional technical information in the project's repository. Next, in Table 3 it is described the application file-system, and in Table 4 it is detailed information about the experiment capture data files.

Table 3: **The VCBot file-system**

(The *src* folder)

Folder	**Description**
data	Storage for configuration parameters, models, datasets, and experiments.
GUI	Python classes for the graphical user interface.
images	Images loaded by the graphical user interface.
lib	NRL dynamic library compiled in the host platform.
tools	Python scripts for processing data.

(The *data* folder)

Folder	**Description**
config	Includes human readable (HR) property files which are passed to the NRL back-end to specify the model parameters and the dataset association.
dataset	Includes folders named as the datasets. Each folder contains a HR *dataset.d* property file indicating information about the number of primitives contained, the sampling period, and the length of the primitives. Data for each primitive is HR, stored in comma-separated values (CSV) files. The naming convention for data is primitive_I_S.*csv*. Where $I \in [0, k)$ is an integer identification for the primitive among k primitives, and $S \in [0, n_I)$ is the number of samples available for primitive I. Although in VCBot it is not possible to capture several samples from the same primitive, NRL allows such possibility in training.
experiment	Includes folders named as the models. Experiments are registered inside these folders named after the date and time stamps of recording. The HR *experiment.d* property file indicates information about the parameters selected for the experiment. Data (see Table 2) is recorded in Python's *numpy* package file format (*npy* extension), which is not HR. However, in VCBot's *Analysis* tab the user can export the experiment data to CSV format.
model	NRL dynamic library compiled in the host platform.

Table 4: **Experiment data files**

File	Description
cur_pos	n vectors $\in \mathbb{R}^2$ of joint positions of the end-effector from emergent behavior, resulting from the robot and the human actions over the end-effector.
tgt_pos	n vectors $\in \mathbb{R}^2$ of the robot intended behavior in the joint space.
hum_pos	n vectors $\in \mathbb{R}^2$ of the human intended behavior in the joint space (captured from the mouse interface).
hum_int	n position boolean array for the event *control* key pressed for interaction.
elbo	n vectors $\in \mathbb{R}^3$ containing information relative to optimal negative evidence lower bound (N-ELBO), reconstruction error from the posterior distribution, and Kullback-Leibler divergence between the prior and posterior distributions, per experiment time step. Since the first *window* time steps are not available for postdiction, NRL sets them to zero.
states	The network internal states concatenated by layers. For example, for the model MacawBot (Low layer: 40 d units, 10 z units; High layer: 10 d units, 1 z unit) n row vector $\in \mathbb{R}^{250}$ are stored per time step, containing the following information: $state_t$ = [Low layer: [prior latent states (40 h, 40 d, 4 μ, 4 log σ, 4 σ, 4 ϵ, 4 z), posterior latent states (40 h, 40 d, 4 μ, 4 log σ, 4 σ, 4 ϵ, 4 z)], High layer: [prior latent states (10 h, 10 d, 1 μ, 1 log σ, 1 σ, 1 ϵ, 1 z), posterior latent states (10 h, 10 d, 1 μ, 1 log σ, 1 σ, 1 ϵ, 1 z)]]. The description of the state variables is given in Table 2.

Acknowledgements

This project was fully funded by the Okinawa Institute of Science and Technology Graduate University (OIST), and developed in the Cognitive Neurorobotics Research Unit (CNRU), which is located in 1919-1, Tancha, Onna, Kunigami District, Okinawa 904-0495.

Valuable feedback was obtained from CNRU researchers which added positively to the final results achieved. A special recognition is done to Munenori Takaku and Siqing Hou for contributing to the multi-platform portability of the project.

References

[1] W. Khalil and E. Dombre, *Modeling, Identification and Control of Robots*. Bristol, PA, USA: Taylor & Francis, Inc., 3rd ed., 2002.

[2] H. F. Chame, A. Ahmadi, and J. Tani, "Towards hybrid primary intersubjectivity: a neural robotics library for human science," 2020.

[3] A. Ahmadi and J. Tani, "A novel predictive-coding-inspired variational rnn model for online prediction and recognition," *Neural computation*, vol. 31, no. 11, pp. 2025–2074, 2019.

[4] H. F. Chame and J. Tani, "Cognitive and motor compliance in intentional human-robot interaction," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020. (in press).

[5] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.