MCC Modular Cerebellar Circuit: Technical Documentation *

Marie Claire Capolei, Silvia Tolu, Henrik Hautop Lund

{macca,stolu,hhl}@elektro.dtu.dk

Automation and Control Group, Department of Electrical Engineering,

Technical University of Denmark, Denmark

Contents

*This work has received funding from the EU-H2020 Framework Program for Research and Innovation under the specific grant agreement No. 785907 (Human Brain Project SGA2)

1 Introduction

The Modular Cerebellar Circuit (***MCC***) is an algorithm that learns online a pattern trough incremental learning, and elaborates an adaptive correction for the control input in order to reduce the error signal. The **MCC** design replicates the functionality, learning, modularity, and synaptic organization of the biological cerebellar-circuit, a neural structure located at the back of the brain, through the combination of machine learning, artificial neural networks, and computational neuroscience techniques. The network weights are defined by non-linear and multidimensional learning functions that mimic the cerebellar synaptic plasticities, as proposed by [1], [2]. Moreover, the biologically plausible weighting kernel together with the layered structure of the cerebellar network result beneficial to constrains the effects of perturbations and nonlinearities.

The **MCC** has been exploited to control a robotic plant, in order to test specific hypothesis regarding the cerebellum, and to improve the performance of the complex robotic system while it is interacting with unknown and dynamic environment [3], [4].

The technical manual includes the list of all the class elements (Section 2), and an example of script file (Section 3).

2 Elements of the MCC Class

2.1 Necessary variables

These variables need to be initialized in the script.

n_uml (*integer*)
N number of unit learning machine.

n_ccm (*integer*)
M number of local specialized units, i.e. cerebellar canonical circuit.

n_input_mf (*integer*)
2N number of sensory signals input through the MF mossy fibers.

2.2 Optional Variables

Calling these variable in the script will override their default value.

name (*string*, Default = 'Cerebellum')
Name of the MCC Modular Cerebellar Circuit.

rfs_print (*integer*, Default = 1)
1 (0) to enable (disable) the visualization of the G number of local linear models created by the LWPR.

signal_normalization (*integer*, Default = 1)
1 (0) to (not) normalize the signal.

PC_on (*bool*, Default = True)
True (False) to (not) include the Purkinje cells contribution to the Deep Cerebellar nuclei.

IO_on (*bool*, Default = True)
True (False) to (not) include the inferior olive contribution to the Deep Cerebellar nuclei.

MF_on (*bool*, Default = True)
True (False) to (not) include the mossy fibers contribution to the Deep Cerebellar nuclei.

2.2.1 Debug Variables

debug_update (*integer*, Default = 0)
1 (0) to enable (disable) the print of the information about the update process.

debug_prediction (*integer*, Default = 0)
1 (0) to enable (disable) the print of the information about the prediction process.

2.3 Signals

x_MF (*float 1-dimensional array*, $1 \times n\_input\_mf$)
Mossy fibers sensory signals,

$$x\_MF = \mathbf{x}^{mf} = \left[x_1^{mf}(t),\ \dots\ ,\ x^{mf}_{n\_input\_mf}\right].$$

x_MF_teach (*float 1-dimensional array*, $1 \times n\_uml$)
$\mathbf{x}^{tot}_n(t-1)$ signal mapped in the granular layer, e.g. signal learned by the LWPR,

$$x\_MF\_teach = \mathbf{x}^{tot} = \left[x_1^{tot}(t-1),\ \dots\ ,\ x^{tot}_{n\_uml}(t-1)\right].$$

x_MF_DCN (*float 2-dimensional array*, $n\_uml \times n\_ccm$)
$\mathbf{x}^{mf-dcn}$ mossy fibers contribution to the deep cerebellar nuclei activity,

$$x\_MF\_DCN = \mathbf{x}^{mf-dcn} = \left[\left[x^{mf-dcn}_{uml_1,ccm_1}, \dots, x^{mf-dcn}_{uml_1,ccm_M}\right], \dots, \left[x^{mf-dcn}_{uml_N,ccm_1}, \dots, x^{mf-dcn}_{uml_N,ccm_M}\right]\right].$$

x_PF (*float 1-dimensional array*, $1 \times n\_output\_pf$)
Signal predicted by the LWPR,

$$x\_PF = \hat{\mathbf{x}}^{pf} = \left[\hat{x}_1^{pf},\ \dots\ ,\ \hat{x}^{pf}_{n\_uml}\right].$$

x_PC (*float 2-dimensional array*, $n\_uml \times n\_ccm$)
$\mathbf{x}^{pc}$ Purkinje cells prediction modulated by the Inferior Olive,

$$x\_PC = \mathbf{x}^{pc} = \left[\left[x^{pc}_{uml_1,ccm_1}, \dots, x^{pc}_{uml_1,ccm_M}\right], \dots, \left[x^{pc}_{uml_N,ccm_1}, \dots, x^{pc}_{uml_N,ccm_M}\right]\right].$$

x_PC_norm (*float 2-dimensional array*, $n\_uml \times n\_ccm$)
Normalized Purkinje cells contribution.

x_PC_DCN (*float 2-dimensional array*, $n\_uml \times n\_ccm$)
$\mathbf{x}^{pc-dcn}$ Purkinje cells contribution to the deep cerebellar nuclei activity,

$$x\_PC\_DCN = \mathbf{x}^{pc-dcn} = \left[\left[x^{pc-dcn}_{uml_1,ccm_1}, \dots, x^{pc-dcn}_{uml_1,ccm_M}\right], \dots, \left[x^{pc-dcn}_{uml_N,ccm_1}, \dots, x^{pc-dcn}_{uml_N,ccm_M}\right]\right].$$

x_IO (*float 2-dimensional array*, $n\_uml \times n\_ccm$)
$\mathbf{x}^{IO}$ inferior olive signals,

$$x\_IO = \mathbf{x}^{io} = \left[\left[x^{io}_{uml_1,ccm_0}, \dots, x^{IO}_{uml_0,ccm_M}\right], \dots, \left[x^{IO}_{uml_N,ccm_0}, \dots, x^{IO}_{uml_N,ccm_M}\right]\right].$$

x_IO_DCN (*float 2-dimensional array*, $n\_uml \times n\_ccm$)
$\mathbf{x}^{io-dcn}$ inferior olive contribution to the deep cerebellar nuclei activity,

$$x\_IO\_DCN = \mathbf{x}^{io-dcn} = \left[\left[x^{io-dcn}_{uml_1,ccm_1}, \dots, x^{io-dcn}_{uml_1,ccm_M}\right], \dots, \left[x^{io-dcn}_{uml_N,ccm_1}, \dots, x^{io-dcn}_{uml_N,ccm_M}\right]\right].$$

x_DCN (*float 1-dimensional array*, $1 \times n\_uml$)
$\Delta\mathbf{x}^{dcn}$ deep cerebellar nuclei output signal,

$$x\_DCN = \Delta\mathbf{x}^{dcn}\left[\Delta x_1^{dcn},\ \dots\ ,\ \Delta x^{dcn}_{n\_uml}\right].$$

2.3.1 Definition of the Signals Boundaries

range_symmetry (*2-dimensional array*, 1×2, Default = 1)
Matrix describing if the boundaries of the inferior olive signal and the deep cerebellar nuclei are 1 (0) symmetric (asymmetric),

$$range\_symmetry = \left[symmetry^{io}, symmetry^{dcn}\right].$$

range_IO (*3-dimensional array*, $n\_uml \times n\_cmm \times 2$, Default = $[0., \pi]$)
Matrix describing the boundaries of the inferior olive signal per each ccm,

$$range\_IO = [\left[\left[min_{ccm_0,uml_0}, max_{ccm_0,uml_0}\right], ..., \left[min_{ccm_M,uml_0}, max_{ccm_M,uml_0}\right]\right], ..., \left[\left[min_{ccm_0,uml_N}, max_{ccm_0,uml_N}\right], ..., \left[min_{ccm_M,uml_N}, max_{ccm_M,uml_N}\right]\right]]$$

.

range_signal (*2-dimensional array*, $n\_uml \times 2$, Default = $[0., \pi]$)
Matrix describing the boundaries of the deep cerebellar nuclei signal per each uml,

$$range\_signal = \left[\left[min_{uml_0}, max_{uml_0}\right], ..., \left[min_{uml_N}, max_{uml_N}\right]\right]$$

.

2.4 Synaptic Plasticity

2.4.1 Granular layer: the Locally Weighted Projection Regression Algorithm

Parameters described in the Locally Weighted Projection Regression (LWPR) [5] user manual [1]. Calling these variable in the script before the ***create_model()*** function will override their default value.

init_D_gr (*float*, Default = 0.5)
Initial values of distance metric.

init_alpha_gr (*float*, Default = 100)
Distance Metric initial learning rate.

w_gen_gr (*float*, Default = 0.4)
Weight activation threshold.

diag_only_gr (*bool*, Default = bool(1))
Diagonal Distance Metric.

update_D_gr (*bool*, Default = bool(0))
Distance Metric receptive fields update.

meta_gr (*bool*, Default = bool(0))
Distance Metric second order adaptation.

[1]Locally Weighted Projection Regression web-site:
https://homepages.inf.ed.ac.uk/svijayak/software/LWPR/

init_lambda_gr (*float*, Default = 0.9)
Initial forgetting factor (commented at the moment).

tau_lambda_gr (*float*, Default = 0.5)
Annealing constant for the forgetting factor (commented at the moment).

final_lambda_gr (*float*, Default = 0.995)
Final forgetting factor (commented at the moment).

w_prune_gr (*float*, Default = 0.95)
Weight prune threshold.

meta_rate_gr (*float*, Default = 0.3)
Second order learning rate (commented at the moment).

add_threshold_gr (*float*, Default = 0.95)
Regression threshold (commented at the moment).

kernel_gr (*string*, Default = 'Gaussian')
Receptive field activation function (commented at the moment).

2.4.2 Synaptic weights

w_gr (*float N-dimensional array*, $n\_uml \times G$)
w^{gr} synaptic weight of the granular cells. The dimension depends on the G number of local linear models created by the LWPR .

w_pf_pc (*float 2-dimensional array*, $n\_uml \times n\_ccm$)
w^{pf-pc} synaptic weight parallel fibers-Purkinje cells connection,

$$w\_pf\_pc = \mathbf{w}^{pf-pc} = \left[\left[w^{pf-pc}_{uml_1,ccm_1}, ..., w^{pf-pc}_{uml_1,ccm_M}\right], ..., \left[w^{pf-pc}_{uml_N,ccm_1}, ..., w^{pf-pc}_{uml_N,ccm_M}\right]\right],$$

the weight increment is store in the variable **delta_w_pf_pc**.

w_pc_dcn (*float 2-dimensional array*, $n\_uml \times n\_ccm$)
w^{pc-dcn} synaptic weights Purkinje cells-deep cerebellar nuclei connection,

$$w\_pc\_dcn = \mathbf{w}^{pc-dcn} = \left[\left[w^{pc-dcn}_{uml_1,ccm_1}, ..., w^{pc-dcn}_{uml_1,ccm_M}\right], ..., \left[w^{pc-dcn}_{uml_N,ccm_1}, ..., w^{pc-dcn}_{uml_N,ccm_M}\right]\right],$$

the weight increment is store in the variable **delta_w_pc_dcn**.

w_io_dcn (*float 2-dimensional array*, $n\_uml \times n\_ccm$)
w^{io-dcn} synaptic weight inferior olive-deep cerebellar nuclei connection,

$$w\_io\_dcn = \mathbf{w}^{io-dcn} = \left[\left[w^{io-dcn}_{uml_1,ccm_1}, ..., w^{io-dcn}_{uml_1,ccm_M}\right], ..., \left[w^{io-dcn}_{uml_N,ccm_1}, ..., w^{io-dcn}_{uml_N,ccm_M}\right]\right],$$

the weight increment is store in the variable **delta_w_io_dcn**.

w_mf_dcn (*float 2-dimensional array*, $n\_uml \times n\_ccm$)
w^{mf-dcn} synaptic weight mossy fibers-deep cerebellar nuclei connection,

$$w\_mf\_dcn = \mathbf{w}^{mf-dcn} = \left[\left[w^{mf-dcn}_{uml_1,ccm_1}, ..., w^{mf-dcn}_{uml_1,ccm_M}\right], ..., \left[w^{mf-dcn}_{uml_N,ccm_1}, ..., w^{mf-dcn}_{uml_N,ccm_M}\right]\right],$$

the weight increment is store in the variable **delta_w_mf_dcn**.

2.4.3 Learning parameters

alpha_PF_PC (*float 2-dimensional array*, *n_uml* × *n_ccm*, Default = 1.)
Decaying factor of the parallel fibers-Purkinje cells connection [2],

$$alpha\_PF\_PC = \left[\left[\alpha^{pf-pc}_{uml_1,ccm_1}, ..., \alpha^{pf-pc}_{uml_1,ccm_M}\right], ..., \left[\alpha^{pf-pc}_{uml_N,ccm_1}, ..., \alpha^{pf-pc}_{uml_N,ccm_M}\right]\right].$$

ltp_PF_PC_max (*float 2-dimensional array*, *n_uml* × *n_ccm*, Default = $10.^{-3}$)
Maximum long term potentiation factor of the parallel fibers-Purkinje cells connection [2],

$$ltp\_PF\_PC\_max = \left[\left[ltp^{pf-pc}_{uml_1,ccm_1}, ..., ltp^{pf-pc}_{uml_1,ccm_M}\right], ..., \left[ltp^{pf-pc}_{uml_N,ccm_1}, ..., ltp^{pf-pc}_{uml_N,ccm_M}\right]\right].$$

ltd_PF_PC_max (*float 2-dimensional array*, *n_uml* × *n_ccm*, Default = $10.^{-3}$)
Maximum long term depression factor of the parallel fibers-Purkinje cells connection [2],

$$ltd\_PF\_PC\_max = \left[\left[ltd^{pf-pc}_{uml_1,ccm_1}, ..., ltd^{pf-pc}_{uml_1,ccm_M}\right], ..., \left[ltd^{pf-pc}_{uml_N,ccm_1}, ..., ltd^{pf-pc}_{uml_N,ccm_M}\right]\right].$$

alpha_PC_DCN (*float 2-dimensional array*, *n_uml* × *n_ccm*, Default = 1.)
Decaying factor of the Purkinje cells-deep cerebellar nuclei connection [2],

$$alpha\_PC\_DCN = \left[\left[\alpha^{pc-dcn}_{uml_1,ccm_1}, ..., \alpha^{pc-dcn}_{uml_1,ccm_M}\right], ..., \left[\alpha^{pc-dcn}_{uml_N,ccm_1}, ..., \alpha^{pc-dcn}_{uml_N,ccm_M}\right]\right].$$

ltp_PC_DCN_max (*float 2-dimensional array*, *n_uml* × *n_ccm*, Default = $10.^{-3}$)
Maximum long term potentiation factor of the Purkinje cells-deep cerebellar nuclei connection [2],

$$ltp\_PC\_DCN\_max = \left[\left[ltp^{pc-dcn}_{uml_1,ccm_1}, ..., ltp^{pc-dcn}_{uml_1,ccm_M}\right], ..., \left[ltp^{pc-dcn}_{uml_N,ccm_1}, ..., ltp^{pc-dcn}_{uml_N,ccm_M}\right]\right].$$

ltd_PC_DCN_max (*float 2-dimensional array*, *n_uml* × *n_ccm*, Default = $10.^{-3}$)
Maximum long term depression factor of the Purkinje cells-deep cerebellar nuclei connection [2],

$$ltd\_PC\_DCN\_max = \left[\left[ltd^{pc-dcn}_{uml_1,ccm_1}, ..., ltd^{pc-dcn}_{uml_1,ccm_M}\right], ..., \left[ltd^{pc-dcn}_{uml_N,ccm_1}, ..., ltd^{pc-dcn}_{uml_N,ccm_M}\right]\right].$$

alpha_MF_DCN (*float 2-dimensional array*, *n_uml* × *n_ccm*, Default = 1.)
Decaying factor of the mossy fibers-deep cerebellar nuclei connection [2],

$$alpha\_MF\_DCN = \left[\left[\alpha^{mf-dcn}_{uml_1,ccm_1}, ..., \alpha^{mf-dcn}_{uml_1,ccm_M}\right], ..., \left[\alpha^{mf-dcn}_{uml_N,ccm_1}, ..., \alpha^{mf-dcn}_{uml_N,ccm_M}\right]\right].$$

ltp_MF_DCN_max (*float 2-dimensional array*, $n\_uml \times n\_ccm$, Default = $10.^{-3}$)
Maximum long term potentiation factor of the mossy fibers-deep cerebellar nuclei connection [2],

$$ltp\_MF\_DCN\_max = \left[\left[ltp^{mf-dcn}_{uml_1,ccm_1}, ..., ltp^{mf-dcn}_{uml_1,ccm_M}\right], ..., \left[ltp^{mf-dcn}_{uml_N,ccm_1}, ..., ltp^{mf-dcn}_{uml_N,ccm_M}\right]\right].$$

ltd_MF_DCN_max (*float 2-dimensional array*, $n\_uml \times n\_ccm$, Default = $10.^{-3}$)
Maximum long term depression factor of the mossy fibers-deep cerebellar nuclei connection [2],

$$ltd\_MF\_DCN\_max = \left[\left[ltd^{mf-dcn}_{uml_1,ccm_1}, ..., ltd^{mf-dcn}_{uml_1,ccm_M}\right], ..., \left[ltd^{mf-dcn}_{uml_N,ccm_1}, ..., ltd^{mf-dcn}_{uml_N,ccm_M}\right]\right].$$

alpha_IO_DCN (*float 2-dimensional array*, $n\_uml \times n\_ccm$, Default = 1.)
Decaying factor of the inferior olive-deep cerebellar nuclei connection [6],

$$alpha\_IO\_DCN = \left[\left[\alpha^{io-dcn}_{uml_1,ccm_1}, ..., \alpha^{io-dcn}_{uml_1,ccm_M}\right], ..., \left[\alpha^{io-dcn}_{uml_N,ccm_1}, ..., \alpha^{io-dcn}_{uml_N,ccm_M}\right]\right].$$

mtp_IO_DCN_max (*float 2-dimensional array*, $n\_uml \times n\_ccm$, Default = $10.^{-3}$)
Potentiation modulating term of the inferior olive-deep cerebellar nuclei connection [6],

$$mtp\_IO\_DCN\_max = \left[\left[mtp^{io-dcn}_{uml_1,ccm_1}, ..., mtp^{io-dcn}_{uml_1,ccm_M}\right], ..., \left[mtp^{io-dcn}_{uml_N,ccm_1}, ..., mtp^{io-dcn}_{uml_N,ccm_M}\right]\right].$$

mtd_IO_DCN_max (*float 2-dimensional array*, $n\_uml \times n\_ccm$, Default = $10.^{-3}$)
Depression modulating term of the inferior olive-deep cerebellar nuclei connection [6],

$$mtd\_IO\_DCN\_max = \left[\left[mtd^{io-dcn}_{uml_1,ccm_1}, ..., mtd^{io-dcn}_{uml_1,ccm_M}\right], ..., \left[mtd^{io-dcn}_{uml_N,ccm_1}, ..., mtd^{io-dcn}_{uml_N,ccm_M}\right]\right].$$

2.5 Functions

create_model()

Initialize the LWPR with the parameters defined in section 2.4.1. The function have to be call after the initialization of the cerebellum. Call the class object ***model*** to access directly to the LWPR parameters.

update_model()

Update the LWPR model given the x_MF and the x_MF_teach (Section 2.3).

prediction(error)

This function output the deep cerebellar nuclei contribution given the input error signal,

$$error_{n\_ccm \times n\_uml} = \left[\mathbf{e}_{ccm_1}, ..., \mathbf{e}_{ccm_N}\right] = \left[\left[e_{uml_1,ccm_1}, ..., e_{uml_N,ccm_1}\right], ..., \left[e_{uml_1,ccm_N}, ..., e_{uml_N,ccm_N}\right]\right];$$

3 Example Script File

In this section, an example of usage is presented. The script intends to use the class to predict the next control signal and add adaptive correction. Before starting, it is important to make sure your computer has python2.7 and the LWPR Locally Weighted Projection Regression packages installed. To install the class run the '*__init__.py*' script first.

```
Define the signals range for both IO and DCN
/* Define all the necessary variable */
n_uml = 3
n_ccm = 1
n_input_mf = 9

/* Initialize the cerebellum class */
cerebellum = MCC(n_uml, n_ccm, n_input_mf)

/* Override variables */
cerebellum.name = "test model"
cerebellum.range_IO = io_range
cerebellum.range_signal = signal_range
cerebellum.init_D_gr = 10.

/* Initialize the LWPR */
cerebellum.create_model()

for $k = 0$, samples do
    /* Get the current sensory signal $x^{mf}(k)$ */
    cerebellum.x_MF = $[x_1^{mf}(k), \dots, x_{n\_input\_mf}^{mf}(k)]$
    /* Get the latest signal $x^{tot}(k-1)$ */
    cerebellum.x_MF_teach = $[x_1^{tot}(k-1), \dots, x_{n\_uml}^{tot}(k-1)]$
    /* Update the LWPR */
    cerebellum.update_model()

    /* Compute the error given the current state information */
    /* Get the cerebellar contribution */
    $\Delta\mathbf{x}^{dcn}$ = cerebellum.prediction(error)

end
```

Algorithm 1: Cerebellum class example script

References

[1] N. R. Luque, J. A. Garrido, R. R. Carrillo, E. D'Angelo, and E. Ros, "Fast convergence of learning requires plasticity between inferior olive and deep cerebellar nuclei in a manipulation task: A closed-loop robotic simulation", *Frontiers in computational neuroscience*, vol. 8, p. 97, 2014.

[2] J. A. Garrido Alcazar, N. R. Luque, E. D'Angelo, and E. Ros, "Distributed cerebellar plasticity implements adaptable gain control in a manipulation task: A closed-loop robotic simulation", *Frontiers in neural circuits*, vol. 7, p. 159, 2013.

[3] M. C. Capolei, N. A. Andersen, H. H. Lund, E. Falotico, and S. Tolu, "A cerebellar internal models control architecture for online sensorimotor adaptation of a humanoid robot acting in a dynamic environment", *IEEE Robotics and Automation Letters*, vol. 5, no. 1, pp. 80–87, 2019.

[4] M. C. Capolei, E. Angelidis, E. Falotico, H. H. Lund, and S. Tolu, "A biomimetic control method increases the adaptability of a humanoid robot acting in a dynamic environment", *Frontiers in Neurorobotics*, vol. 13, p. 70, 2019, ISSN: 1662-5218. DOI: 10.3389/fnbot.2019.00070. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnbot.2019.00070.

[5] S. Vijayakumar and S. Schaal, "Locally weighted projection regression: An o (n) algorithm for incremental real time learning in high dimensional space", in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, vol. 1, 2000, pp. 288–293.

[6] N. R. Luque, R. R. Carrillo, F. Naveros, J. A. Garrido, and M. Sáez-Lara, "Integrated neural and robotic simulations. simulation of cerebellar neurobiological substrate for an object-oriented dynamic model abstraction process", *Robotics and Autonomous Systems*, vol. 62, no. 12, pp. 1702 –1716, 2014.