2017 IEEE International Conference on
Autonomous Robot Systems and Competitions (ICARSC)
April 26-28, Coimbra, Portugal

A software framework for the implementation of dynamic neural field control architectures for human-robot interaction

Tiago Malheiro, Estela Bicho, Toni Machado,
Luis Louro, Sergio Monteiro, Paulo Vicente
Department of Industrial Electronics and Centre Algoritmi
University of Minho
Canpus de Azurem, 4800–058 Guimaraes, Portugal
Email: {tmalheiro,estela.bicho,tmachado,llouro,sergio,pvicente}@dei.uminho.pt

Wolfram Erlhagen
Department of Mathematics and Applications
Center for Mathematics
University of Minho, Portugal
Email: wolfram.erlhagen@math.uminho.pt

***Abstract*—Useful and efficient human-robot interaction in joint tasks requires the design of a cognitive control architecture that endows robots with crucial cognitive and social capabilities such as intention recognition and complementary action selection. Herein, we present a software framework that eases the design and implementation of Dynamic Neural Field (DNF) cognitive architectures for human-robot joint tasks. We provide a graphical user interface to draw instances of the robot's control architecture. In addition, it allows to simulate, inspect and parametrize them in real-time. The framework eases parameter tuning by allowing changes on-the-fly and by connecting the cognitive architecture with simulated or real robots. Using the case study of an anthropomorphic robot providing assistance to a disabled person during a meal scenario, we illustrate the applicability of the framework.**

I. Introduction

As robot systems are entering into human everyday life the question of how to design robots capable of acting as socially intelligent assistants is becoming increasingly important [1]–[3]. Useful and efficient human-robot interaction requires that both agents coordinate and synchronize their decisions and actions in any given joint task. In order to decrease the workload of the human needing assistance, the robot should actively contribute to this coordination effort. This means that the robot should exhibit cognitive and social capacities such as action understanding and intention recognition.

In an interdisciplinary effort involving cognitive scientists and roboticists (EU Integrated Project JAST and EU ITN Marie Curie NETT) we have developed an autonomous anthropomorphic robot that integrates in its control architecture known neurocognitive mechanisms supporting human–human collaboration in joint tasks. One key idea is that during observation of an action, a corresponding representation in the observer's motor system is activated that allows the observer to predict the action goal of the partner. During joint action, the representation of the inferred goal together with representations of prior task knowledge may then automatically bias the observer's decision process towards selecting an adequate complementary behaviour (review e.g. [4]).

978-1-5090-6234-8/17/$31.00 ©2017 IEEE

The robot control architecture for human-robot interaction in joint tasks implements such a context-sensitive, i.e. flexible, mapping between action observation and action execution (see Fig. 1). The coordination of actions and decisions among the two agents is modeled as a dynamic process that builds on the continuous integration of input from representations of the inferred goal (Intention Layer) of observed actions (obtained through action simulation), contextual cues (e.g., location of objects in the scene, represented in Objects Memory Layer) and shared task knowledge (e.g., sequence of steps to serve a drink represented in Common Sub-Goals Layer). The representation of the complementary action that gets the strongest support (Action Execution Layer) will win the dynamic competition process among all possible complementary behaviours. As a theoretical framework we have used the Dynamic Neural Field (DNF) approach to robotics [5]. Originally introduced as a simplified mathematical model for pattern formation in neural populations [6], [7], DNFs have been later generalized and applied to the cognitive domain (review see [8]). The architecture of DNFs reflects the hypothesis that strong recurrent interactions in local populations of neurons form a basic mechanism of cortical information processing. These interactions support the existence of self-stabilized representations that allow the cognitive agent for instance to compensate for temporally missing sensory input, or to anticipate future environmental inputs that may inform the decision about a specific goal-directed behaviour.

In summary, the DNF–model of joint action forms a complex dynamical system consisting of a distributed network of reciprocally connected neural populations that integrate and represent in their activation patterns task–relevant information. The model has been experimentally validated in several human–robot joint action scenarios [10]–[13]. For each task scenario, the authors translated the designed model architecture into a C++ language application with all neural fields and inter–field synaptic connections hand coded into the program

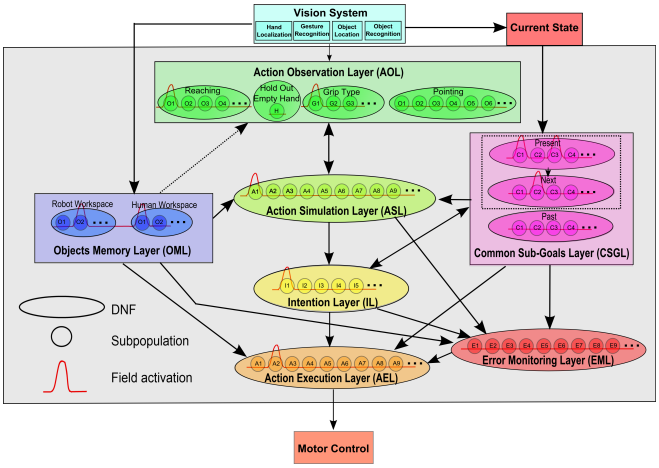


Fig. 1. Schematic view of the cognitive architecture for human-robot interaction in joint action tasks. It implements a flexible mapping from observed actions (layer AOL) onto complementary actions (layer AEL) taking into account the inferred action goal of partner (layer IL), detected errors (layer EML), contextual cues (OML) and shared task knowledge (CSGL). The goal inference capacity is based on motor simulation (layer ASL). (for details see e.g. [9])

(for a mathematical formulation see e.g. [9]). In addition to the time-consuming work that such a translation process requires, it is prone to hidden errors which might get noticed only during the robotics experiments. Furthermore, changes to the system imply to stop, change, compile, debug, and run cycles. This may hinder, slow down and shift away the attention from the development process of the cognitive control architecture. Moreover, both the adaptation of an existing architecture to new HRI tasks and its usage in real robotics experiments by users unfamiliar with DNF theory are limited due to a significant initial learning effort involved. In order to mitigate these burdens, we have developed a framework to ease the designer work and get him/her focused on the design itself and not on the implementation. The framework is composed of two main tools - `Designer` and `Engine` - and some additional `utilities` (e.g. robotic simulation) to help the developer to describe, implement and evaluate joint action models for HRI. In addition, the framework supports the developer in the parameter tuning of the control system by allowing changes at run-time.

In what concerns related frameworks, Neuron [14] is a simulation environment for models of biological and artificial neurons.It allows to specify the dynamic properties of individual neurons and to select the mathematical model governing their dynamics. Brian [15] is a python based simulator for spiking neural networks. It provides an interface for specifying the neuron model which is more flexible and user-friendly than Neuron. However, both frameworks focus on the detailed simulation of neural firing patterns to allow for comparison with real neural data. This level of detail makes it difficult to implement simulations of highly complex cognitive functionalities required for joint action . The Nengo simulator [16] is based on the Neural Engineering Framework (NEF) [17] and provides features to facilitate the development of large scale neural networks to emulated functional activities of the human brain [18]. It is not designed to meet the real-time constraints of robotics applications. Neuron, Brian and Nengo have in common that neural information processing is based on spiking activity of neurons. In contrast, DNF theory postulates neural activation patterns representing the mean activity of neural populations as fundamental information processing units. As a major advantage for a system developer, this level of abstraction favors analytical treatment of a complex dynamical system (e.g., system stability, [5]). Cedar [19] is a DNF based framework for embodied artificial cognitive systems. It has been applied thus far for robotics applications that can be described by a set of continuous variables (e.g., movement direction in object manipulation). In its present version, it cannot be easily scaled to more complex scenarios involving a large number of neural representations that encode also more abstract information (e.g., grasping type, action goal). Here, we present a novel framework for developing highly complex DNF-based cognitive control architectures and test the applicability in a simulated human-robot interaction task.

The remainder of the paper is structured as follows: next, section II presents the developed software framework; section III illustrates its application to a case study in which the robot provides assistance to a disabled person during a meal scenario. The paper ends in section IV with conclusions and an outlook on future work.

II. The Software Framework

Herein, subsection II-A presents the `Designer` application which allows the user to graphically draw an instance of the control architecture for human-robot interaction for a new joint task. Subsection II-B introduces the `Engine` which processes the architecture instance and allows to monitor it. Connected to the `Engine`, there can exist a virtual robot (e.g. a simulator), a real robot, or any other sensing/actuation system. In subsection II-C we provide an overview of a set of `tools/utilities` developed to ease the connection to the robot.

A. Designer

The `Designer` is an application which allows the user/developer to draw an instance of the DNF cognitive architecture by means of a graphical language. Any instance of the DNF cognitive architecture is built from a set of building blocks, such as `neural fields`, `neural populations`, and `inter-field synaptic connections`. These are the fundamental entities that allow to describe the dynamic neural fields hierarchy and their inter-relations. By means of drag-n-drop, the user can layout the envisioned architecture. `Fields` are representations of Dynamic neural fields, allowing the user to configure their properties. `Populations` are representations of the neural populations essential to the DNF joint action model. These present always a child relationship towards a neural field and

might have zero or many input and output connections. The `connections` are representations of the inter-field synaptic projections. Furthermore, each field has a number of parameters and properties which can be configured and selected from the same graphical application. Specifically, the application allows the user to select properties such as the neural field interaction kernel (i.e. weight function), output function type and dynamic reset source. The following parameters can be tuned for the specific neural field function: resting level, time constant, field discretization and kernel parameters. Figure 2 presents a snapshot of the application with an open dynamic cognitive architecture instance. A first main area is the application canvas (center large area). `Layers`, `fields`, `populations` and `inter-field synaptic connections` are specified hereby drag-n-drop entities from the right. Every entity property (excluding hierarchy) can be set in the secondary left area. Properties are grouped by category, so that navigation is easier. The current properties view are attributed to the selected entity in the main area. At the the bottom of the application is located another secondary view: inter-field projections weights. In this view the user can set e.g. all the synaptic weights for the input projections to a specific dynamic neural field.

At any time, the user can run the drawn instance by sending it to the `Engine`. The design specification is serialized and then loaded by the `Engine`. This allows for a quick evaluation of the system. Furthermore, to ease parameter tuning, a fundamental feature was included: any change to parameters is sent automatically to the `Engine` and applied to the running instance immediately. This direct link aids in understanding the effect of a parameter change in run-time without requiring to stop or restart. The user can see the dynamic evolution of the field activation pattern. Although at run-time structural changes are limited (e.g. new fields and populations are not taken into consideration until full reload), most parameters changes are reflected on-the-fly. Resting level, time constant, kernel excitation and width, and synaptic weights are examples of parameters allowed to be changed at run-time without the need to reload the architecture specification or even restart of execution.

B. Engine

The `High Level Cognition Engine` (or simply `Engine`) is the counter-part of the `Designer` Application. It is able to load an architecture specification written by the Designer and execute it according to the Dynamic Neural Field Human-Robot Joint Action model principles, see [9]–[12]. This application is composed of a GUI (see Fig. 3) and a C++ lib core. The model equations are implemented in the core lib which exposes a set of methods allowing not only to control the execution, but also give inputs, get outputs, and change configuration at run-time. The GUI part is responsible for the execution management (start, pause, stop, clear, and restart), inputs/outputs emulation, plotting and results file management. Central to the application is the ability to draw simultaneously several 2D plots at run-time, see Fig. 3. Properties such as field activity (u), inputs (S), resting level (h), and multiple synaptic projections can be draw at a selectable update rate. The plotting allows the user to have feedback on the fields activity evolution over time. Additionally, it supports plotting of one 3D graph of a selected property. The plot runs also in real-time, but it is an external application dealing with the 3D drawing. As previously noted, of high importance for the design tuning, the core allows most of the parameters to be changed at run-time. While the architecture is being executed, parameters such as field time constant, resting level, and interaction kernel amplitude can be changed in the `Designer` (which automatically sends it to the `Engine`) and feedback is provided in real time, e.g. in the plots. This feature has proven to be of high value for the user due to the efficacy (low latency,...) between change and effect. This parameters tuning versatility is not extended to structural changes in design. As thus, this type of changes (e.g. a new neural field, new population) do require the engine to reload the full architecture.

Each designed dynamic of a cognitive architecture instance might have a unique set of inputs/outputs (e.g. sensory information from cameras, actuation like motor primitives). Although the user is able to use the GUI inputs/outputs emulation controls to provide inputs to the architecture and get feedback on activated outputs, connection to the real robot and/or simulator is a must. In order to tackle the sources variability, as information provider and communication interface, we have chosen to implement a feature allowing the user to dynamically load/unload a plugin responsible for such operations. Such plugin can be implemented as a shared library and we named it `TaskConnector`. Because it is a shared library (DLL, Dynamic Link Library) the user is free to use the language of choice to detail how the interface with the robot is implemented. Restriction happens only at the Application Programming Interface (API) which `Engine` expects the DLL to have. Namely, an entry point that will be called for collecting inputs and a second for outputs. This allows to abstract the `Engine` from the sensors/actuators connections management and thus share it amongst projects.

This `TaskConnector` mechanism allows one to effectively switch between IO interaction by means of reloading of the DLL. In our project, we use it to switch between connecting the `Engine` to the real robot or to the simulator. Each `TaskConnector` has thus the details of how the architecture gathers sensory information from the environment and provides output commands for actuation.

Input/output description starts at the design level. In the `Designer`, the user must create a field entity named "`Perceptual Field`" for the inputs and a second named "`Motor Control`" for the outputs. The `Engine` will search for these when the architecture is loaded and connects them to the `TaskConnector`. The `TaskConnector` must, therefore, be implemented in such a way that it is able to translate sensory information into the "`Perceptual Field`" entity. Furthermore, it should be able to interpret the "`Motor Control`" information and translate it to the proper robot

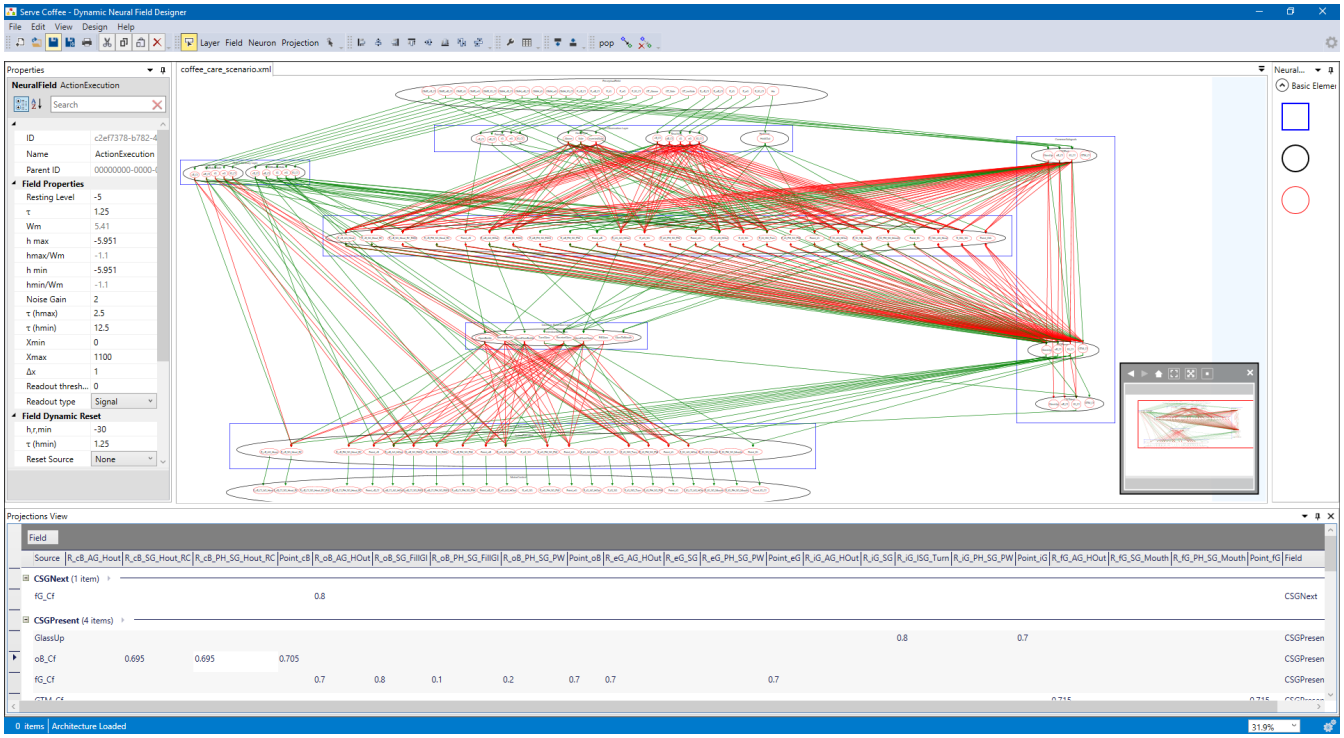


Fig. 2. `Designer`: The application main area is attributed to the design canvas. In this area, the architecture can be graphically detailed, by laying out layers (rectangles), Fields (black outer ellipses), and neural populations (inner red ellipses) and defining their relationship (parent/child). Arrows are representations of synaptic projections between two neural populations. An excitatory projection gets a green line color. On the other side, inhibitory projections gets a red line color.

command.

The `Engine` includes also another feature: `data storage` for post-execution analysis. Field properties input (S), resting level (h), and activity (u) can be stored into a selected file. The user can select the result file update rate. To limit the result file size and required write throughput to disk, the information is stored in binary format. To ease post processing usage of the stored data, we provide a data parser which extracts file contents into multiple files, one per architecture field and stored property. This parser can output in binary format (good for example to load it with MatLab) or tab separated ASCII file, for user readable format.

C. Utilities

The above described framework (`Designer` and `Engine`) is self-contained and the developer can evaluate simple aspects of the robot cognitive system by emulating inputs and analyzing outputs. Nevertheless, for more complex cognitive capabilities, and to actually interact with the environment, other utilities can benefit the evaluation. Namely, a simulation environment and/or an sensor array to gather external influences and actuators to take actions in the robot's environment.

To improve the assessment of the designed architecture under human-robot interaction scenarios, we have developed a scenario in the state of the art simulator VREP [20] including a human and a robot in an assistive task (c.f. Figure 5). The assistive task objective is to help a disabled person to have coffee. Due to the user physical limitations, he/she cannot remove the bottle cap by himself/herself. The robot and human must collaborate in order to achieve the objective. The robot hand dexterity also does not allow it to grasp the bottle cap in order to turn the cap and detach it.

The scenario is accompanied by two plugins which enable robot actions executions and sensory information gathering from the network using the yarp communication framework [21]. Robot actions are fully controlled by the plugin, where trajectories execution and actions such as grasp/release object are given by the connected `Goal Directed Action Server` (GDAS).

The framework is bundled with some other applications and services which allow not only the interconnection of the several involved parties, but provide services such as motion planning. Figure 4 shows the network of applications used in the framework and their interconnections.

As previously described, the user inputs the model architecture into the `designer` (**DyFAD**, `Dynamic Neural Field Architecture Designer` in Fig. 4). The result is serialized and sent to the `Engine` (**HLC Engine**, `High Level Cognition Engine`). This application also gets on-the-fly configuration changes. By means of `TaskConnectors` (see subsection II-B, the `Engine` can be connected to virtual (simulator) or real robots.

GDAS stands for `Goal Directed Action Server`. This application provides a mechanism to translate the selected

Fig. 3. `Engine` with Intention Detection and Action Execution neural fields plots. **S**, **h**, and **u** are the neural field input, resting level, and activation, respectively.

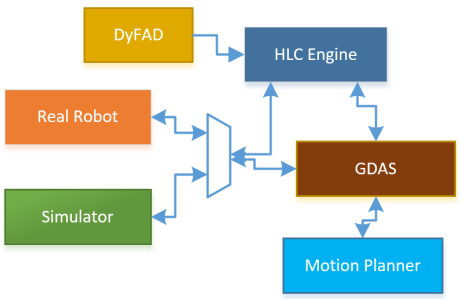


Fig. 4. System Diagram. DyFAD, HLC Engine, and GDAS are the Designer, Engine and Goal directed action server, respectively.

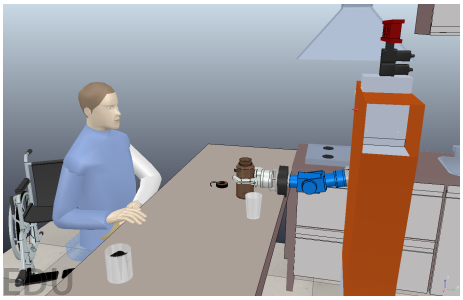


Fig. 5. Assistive Scenario: Drinking Task. (VREP 3.3.0 rev 1)

Goal directed action of the robot - in the Action Execution Layer - into chains of motor primitive. This translation also includes building of motion planner requests and management of its execution.

The **motion planner** is implemented as a service provider which upon a plan request, computes a human-like trajectory for the robot manipulator. This planner is based on a global planning method in posture space that allows to integrate optimization principles derived from experiments with humans [22].

III. Results: The Case Study of a Home Care Scenario

As an usage example of the described framework, consider the scenario introduced above (see Fig 5). In the assistive task, the robot must support a disabled person to drink coffee. Due to both actors limitations, the robot's control challenge is to pro-actively collaborate in an efficient way (here we give focus only on the cognitive part) so that its actions properly complement those of the human. For simplicity, we build a dynamic cognitive architecture instance with a small set of possible human and robot actions. Although this limits robot's possible actions, it does not hinder the objective here of presenting the usage of the framework or suitability to deal with the Joint Action model.

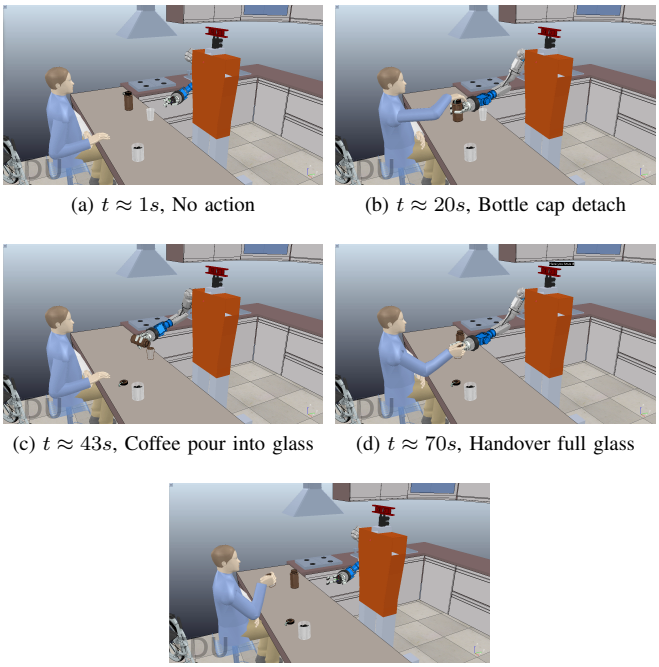
Taking this into consideration, one can start to draft the dynamic cognitive architecture instance by drawing the required DNFs (black outer ellipses, see Figure 2) which endow the robot with the capabilities of action simulation, goal inference, intention detection and complementary action selection. We will not consider error detection. Following it, neural fields can be populated with representations (red inner ellipses) of, e.g., the objects in consideration. Furthermore, inter-field synaptic projections can be added by means of arrows. By setting the synaptic weight to each projection, the arrow line color will change to indicate an excitation (green) or an inhibition (red) projection. This allows to specify the dynamic cognitive architecture instance targeted for the assistive task proposed. By adding inputs and outputs, we can define the link to the sensory and actuation systems. For that, a "`Perceptual Field`" and a "`Motor Control`" field entity should be included in the design. The `Engine` will use them to gather

external inputs to the cognitive system and provide outputs (in this case, goal directed actions) to the robot.

In this study we use a simulated scene to close the loop between the cognitive system and the environment where robot and human are embedded. Figure 5 presents a snapshot of the scenario. As can be seen, both robot and human present physical limitations which prevent each one from completing the task alone. Opening the bottle does require the robot to hold the bottle while the human detaches the bottle cap. Figure 6 presents a snapshot sequence of the interaction at key points from the recorded simulation environment.

At the beginning, no action is taken either from the robot or human. The cognitive system is prepared to, as best as possible, serve the human. In the lack of a user request, the robot will take the initiative and start to prepare the drink. In a more complex setting, the trigger for the robot to serve, e.g., water, could come from several sources. Namely, a user's medication timetable, a program to support the user maintaining recommended hydration level, etc. In this work, the designer configured the robot cognitive system to be pro-active and fast in taking actions even when the human does nothing. As a consequence, it decides to open the bottle (Figure 7e, $t \approx 3.6s$). This can be explained taking into consideration the context. The ultimate goal is to serve a drink to the human user. To that end, several sub-goals must be accomplished beforehand. Namely, grasp the glass, place the glass up, open bottle, and fill glass. These sub-goals are represented in the Common Sub-Goals layer (Figure 7c and 7d). When the system is presented with the closed bottle and empty glass information (consider $t \approx 0s$), it quickly evolves self-stabilized activation patterns in the Object Memory Layer (OML) (see Figure 7a and 7b) and Common Sub-Goals Layer. External information comes from the perceptual subsystem. Since "Glass Up" sub-goal is already achieved (positive activation pattern in "Glass Up" population in CSGPast), the only currently supported sub-goals is open bottle (note the positive activation pattern in CSGPresent from start up to $t \approx 20s$). This information together with the activation in the Object Memory Layer leads to positive excitation in some populations in the Action Execution Layer (AEL) (see Figure 7e). The complementary goal directed actions (GDA) 'R_cB_SG_Hout_RC', 'R_cB_PH_SG_HOut_RC' and 'Point_cB' compete for overt expression because they are possible actions the robot could take which eventually would lead it to satisfy the present sub-goal. 'Point_cB' would win the competition if the closed bottle was in the human workspace, leading the robot to request the human to handover it. It is, thus, the Object Memory Layer projection which makes the robot to decide to reach for the closed bottle with side grip and hold out until the human removes the bottle cap.

When the perceptual system detects the bottle is open ($t \approx 20s$), the Common Sub-Goals layer is updated accordingly. 'Open Bottle' population in CSGPresent loses its positive activation pattern in benefit of 'Fill Glass'. Due to the contextual information, several goal directed actions are biased in AEL. Namely, 'handover open bottle', 'handover empty glass', and 'reach open bottle and fill glass'. Because both the bottle and the glass are within robot's reach, it is more efficient for the overall interaction that the robot reaches for the open bottle, grasps it and fills the glass. The alternative would be for the robot to handover the bottle and glass to human having him/her to pour the beverage. Should the user have requested for the bottle and glass, the robot would try to handover them. Last, having a full glass ($t \approx 43s$) will trigger the evolution of a positive activation pattern in 'Glass to Mouth' population in CSGPresent field. This bias equally both 'handover full glass' ('R_fG_AG_HOut') and 'point to full glass' ('Point_fG') populations in AEL. Nevertheless, the handover action has a competitive advantage over pointing due to the additional information coming from the OML signaling the full glass in the robot's workspace and not in human's workspace.



(a) $t \approx 1s$, No action (b) $t \approx 20s$, Bottle cap detach

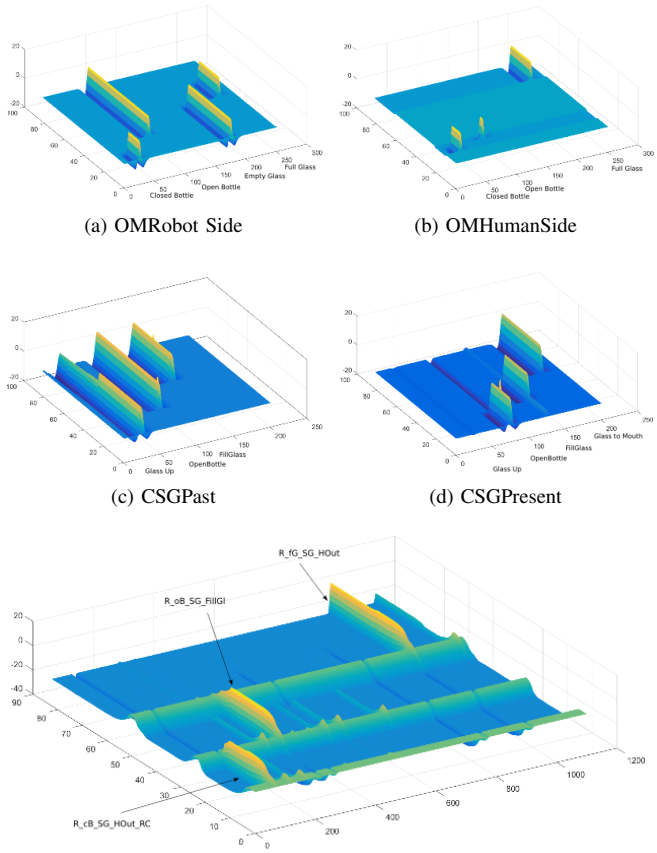
(c) $t \approx 43s$, Coffee pour into glass (d) $t \approx 70s$, Handover full glass

(e) $t \approx 75s$, Glass to mouth

Fig. 6. Snapshot sequence showing the interaction at keys points. The snapshots were taken from the simulated interaction scenario. Full video can be seen in http://marl.dei.uminho.pt/public/videos/CareAssistiveICARSC2017.html.

IV. Conclusion

We have presented a software frameworks that eases the design and implementation of Dynamic Neural Field (DNF) cognitive control architectures for human-robot interaction in joint tasks. In a case study, consisting of a robot providing assistance to a disabled person aiming to drink coffee, we have illustrated its usability and have demonstrated the key requirements that we identified in section I. Ease of implementation is achieved by using a graphical drag-n-drop interface for importing components like `layer`, `dynamic field`, `neural populations` and `inter-field synaptic`



(a) OMRobot Side (b) OMHumanSide

(c) CSGPast (d) CSGPresent

(e) Action Execution

Fig. 7. 3D view of several several dynamic neural fields activation over time. Samples taken with a frequency of 50Hz from a system running at 200Hz.

`connections`. The framework allows connecting the cognitive architecture with simulated or physical devices such as sensors and the anthropomorphic robot. Parameters can be changed on the fly, which facilitates tunning of robot decisions and actions in time and space. Instances of the cognitive architecture can be built, simulated, inspected and parametrized in real time, enabling quick development and validation.

Future work concerns implementing features which allow user to specify learning rules for inter-field synaptic projections weights auto-tuning [23].

ACKNOWLEDGMENT

The work was funded by Project **NETT**: Neural Engineering Transformative Technologies, EU-FP7 ITN (nr.289146), and by FCT - Fundação para a Ciência e Tecnologia, through the Phd and Posdoc Grants (SFRH/BD/81334/2011 and SFRH/BPD/71874/2010 respectively, financed by POPH-QREN-Type 4.1- Advanced Training, co-funded by the European Social Fund and national funds from MEC), and Project Scope: UID/CEC/00319/2013 together with COMPETE: POCI-01-0145-FEDER-007043.

REFERENCES

[1] T. Fong, I. Nourbakhsh, and K. Dautenhahn, "A survey of socially interactive robots," *Robotics and Autonomous Systems*, vol. 42, no. 3-4, pp. 143–166, 2003.

[2] C. Breazeal, "Social interactions in hri: the robot view," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 34, no. 2, pp. 181–186, 2004.

[3] S. Schaal, "The new robotics-towards human-centered machines," *HFSP Journal*, vol. 1, no. 2, pp. 115–126, 2007.

[4] H. Bekkering, E. R. De Bruijn, R. H. Cuijpers, R. Newman-Norlund, H. T. Van Schie, and R. Meulenbroek, "Joint action: Neurocognitive mechanisms supporting human interaction," *Topics in Cognitive Science*, vol. 1, no. 2, pp. 340–352, 2009.

[5] W. Erlhagen and E. Bicho, "The dynamic neural field approach to cognitive robotics." *Journal of neural engineering*, vol. 3, no. 3, pp. 36–54, sep 2006.

[6] S. Amari, "Dynamics of pattern formation in lateral-inhibitory type neural fields." *Biological Cybernetics*, vol. 27, pp. 77–87, 1977.

[7] H. R. Wilson and J. D. Cowan, "A mathematical theory of the functional dynamics of cortical and thalamic nervous tissue," *Kybernetik*, vol. 13, pp. 55–80, 1973.

[8] G. Schöner, "Dynamical systems approaches to cognition," in *The Cambridge Handbook of Computational Psychology*, R. Sun, Ed. Cambridge University Press, 2008, pp. 101–125.

[9] W. Erlhagen and E. Bicho, "A Dynamic Neural Field Approach to Natural and Efficient Human-Robot Collaboration," in *Neural Fields, Theory and Applications*, S. Coombes, P. beim Graben, R. Potthast, and J. Wright, Eds. Springer Berlin Heidelberg, 2014, pp. 341–365.

[10] E. Bicho, W. Erlhagen, L. Louro, and E. Costa e Silva, "Neuro-cognitive mechanisms of decision making in joint action: A human-robot interaction study," *Human Movement Science*, vol. 30, no. 5, pp. 846–868, oct 2011.

[11] E. Bicho, W. Erlhagen, L. Louro, E. Costa e Silva, R. Silva, and N. Hipólito, "A dynamic field approach to goal inference, error detection and anticipatory action selection in human-robot collaboration," in *New Frontiers in Human-Robot Interaction (Advances in Interaction Studies)*, K. Dautenhahn and J. Saunders, Eds. John Benjamins Publishing Company, 2011, pp. 135–164.

[12] E. Bicho, L. Louro, and W. Erlhagen, "Integrating verbal and nonverbal communication in a dynamic neural field architecture for human-robot interaction," *Frontiers in Neurorobotics*, vol. 4, no. MAY, pp. 1–13, 2010.

[13] R. Silva, L. Louro, T. Malheiro, W. Erlhagen, and E. Bicho, "Combining intention and emotional state inference in a dynamic neural field architecture for human-robot joint action," *Adaptive Behavior*, vol. 24, no. 5, pp. 350–372, 2016.

[14] M. L. Hines and N. T. Carnevale, "The NEURON simulation environment," *The Handbook of Brain Theory and Neural Network*, vol. 2, 2002.

[15] D. F. M. Goodman and R. Brette, "The brian simulator," *Frontiers in Neuroscience*, vol. 3, no. SEP, pp. 192–197, 2009.

[16] T. C. Stewart, B. Tripp, C. Eliasmith, S. TC, T. B, and E. C, "Python scripting in the nengo simulator." *Frontiers in neuroinformatics*, vol. 3, no. March, p. 7, 2009.

[17] C. Eliasmith, "A Unified Approach to Building and Controlling Spiking Attractor Networks," *Neural Computing*, vol. 17, no. 6, pp. 1276–1314, 2005.

[18] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. Dewolf, Y. Tang, and D. Rasmussen, "A Large-Scale Model of the Functioning Brain," Tech. Rep. 6111, 2012.

[19] O. Lomp, M. Richter, S. K. U. Zibner, and G. Schöner, "Developing Dynamic Field Theory Architectures for Embodied Cognitive Systems with cedar," *Frontiers in Neurorobotics*, vol. 10, no. November, p. 14, 2016.

[20] E. Rohmer, S. P. N. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework," in *IEEE International Conference on Intelligent Robots and Systems*, 2013, pp. 1321–1326.

[21] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: Yet another robot platform," *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, pp. 043–048, 2006.

[22] E. Costa e Silva, F. Costa, E. Bicho, and W. Erlhagen, *Nonlinear Optimization for Human-Like Movements of a High Degree of Freedom Robotics Arm-Hand System*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 327–342.

[23] E. Sousa, W. Erlhagen, F. Ferreira, and E. Bicho, "Off-line simulation inspires insight: A neurodynamics approach to efficient robot task learning," *Neural Networks*, vol. 72, no. November, pp. 123–139, 2015.