Available online at www.sciencedirect.com

ScienceDirect

Cognitive Systems Research 50 (2018) 52–66

Cognitive Systems RESEARCH

www.elsevier.com/locate/cogsys

Neuromorphic sensorimotor adaptation for robotic mobile manipulation: From sensing to behaviour

Florian Mirus [a,b,*], Cristian Axenie [a], Terrence C. Stewart [c], Jörg Conradt [a]

[a] *Neuroscientific System Theory Group, Department of Electrical and Computer Engineering, Technical University of Munich, Arcisstrasse 21, 80333 Munich, Germany*

[b] *BMW Group E|E Architecture, Technologies, Parkring 19, 85748 Garching, Germany*

[c] *Centre for Theoretical Neuroscience, University of Waterloo, Waterloo, ON, Canada*

Received 31 March 2017; received in revised form 12 December 2017; accepted 12 March 2018
Available online 29 March 2018

Abstract

We propose a neuromorphic approach to perception, reasoning and motor control using Spiking Neural Networks in mobile robotics. We demonstrate this by using a mobile robotic manipulator solving a pick-and-place task. All sensory data is provided by spike-based silicon retina cameras - eDVS (embedded Dynamic Vision Sensor) - and all reasoning and motor control is implemented in Spiking Neural Networks. For the given scenario, the robot is capable of detecting a sequence of objects blinking at different frequencies, finding one object that is not in the right place of the sequence, picking up this object and moving it to its correct position. Such a scenario demonstrates how to build large-scale networks solving a high-level cognitive task by combining several smaller networks responsible for low-level tasks. Importantly, here we focus only on generating a neural network that is capable of performing the task. This will be the basis of future work using neural network learning algorithms to improve task performance. The long-term goal is to learn sophisticated behaviours by experience while at the same time being able to introduce expert knowledge for intermediate tasks that can be used to initialize the network or to speed up the learning process.

Keywords: Mobile robotics; Spiking Neural Networks; Cognitive robotics; Neuromorphic sensing

1. Introduction

We propose a novel neural controller for a mobile manipulator platform that can adapt its control policy for grasping objects within a visual scene. In this paper, we focus on a simplified pick and sort task. We use our mobile robot to manipulate objects equipped with LED stimuli (see Fig. 1) blinking at different frequencies with one being out of the assumed order. The task is to place the objects such that they are arranged by their blinking frequencies in correct (descending) order. We propose a system which uses a spiking neural substrate for representation and computation, that allows the system to approximate sensorimotor correlations for both basic and complex motion and grasping scenarios. Although sorting is not an essential task for mobile robot manipulation, we use it as a sample scenario for our approach to computation for flexible cognitive robotics. The simplified nature

* Corresponding author at: Neuroscientific System Theory Group Department of Electrical and Computer Engineering, Technical University of Munich, Arcisstrasse 21, 80333 Munich, Germany.

E-mail addresses: florian.mirus@tum.de (F. Mirus), cristian.axenie@tum.de (C. Axenie), tcstewar@uwaterloo.ca (T.C. Stewart), conradt@tum.de (J. Conradt).

URL: http://www.nst.ei.tum.de (J. Conradt).

https://doi.org/10.1016/j.cogsys.2018.03.006
1389-0417/

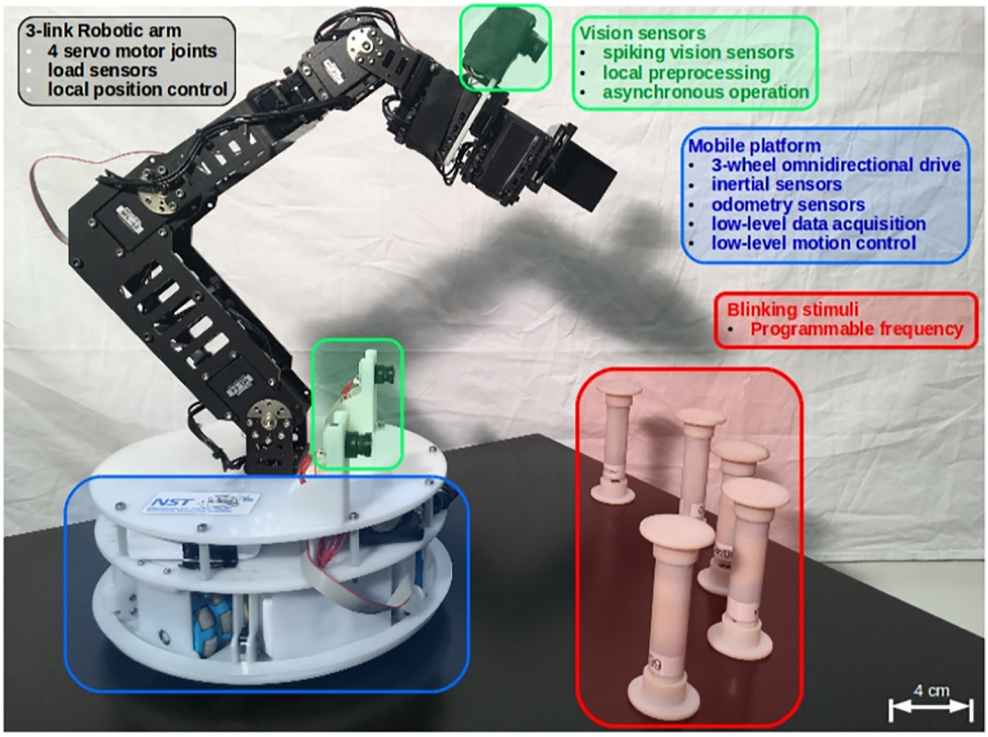

Fig. 1. Robotic platform.

of the task using objects with blinking LEDs, which are easy to detect for event cameras, serves as a test case for our approach of encoding sensory data, implementing perception and motor mappings and dynamically behave in noisy, uncertain environments using spiking neurons as a computing framework. Constructing such a system entirely with simulated neurons gives us two unique advantages. First, the resulting system can be run on energy-efficient neuromorphic hardware. Second, we can use the network that we design here as the starting point for learning from experience (as opposed to traditional neural network solutions, which learn from a blank state). However, in order to generate an initial functioning neural network model that could be used to bootstrap learning, we need a way to program such a network using something similar to traditional engineering programming methods. That is the focus of this paper. The proposed system describes a unified design approach that links low-level sensorimotor data representation with high-level reasoning using a generic computational substrate.

1.1. Related work

Goal-directed movements represent an intermediate level of behaviour that embodies both low-level motor execution and higher-level cognition (Krakauer & Mazzoni, 2011). The richness and variety of learned real-world motor behaviours can be reduced to a small repertoire of tasks that build up from the simple to the complex. Moreover, controlling one's body is the first prerequisite to successful interaction with the environment. Even seemingly simple goal-directed movements are learned. To enable the acquisition, as well as the continuous adaptation of behaviour to changing environmental constraints, modular hierarchical control architectures appear necessary (Butz, Herbort, & Hoffmann, 2007). Looking at neural systems, a decision for an action emerges from competition between different movement plans, which are specified and selected in parallel. One particular implementation of action selection in spiking neurons is the model of the Basal Ganglia in Stewart, Choo, and Eliasmith (2010). For action choices which are based on ambiguous sensory input, neural networks responsible for processing encode alternative spatial motor goals in parallel during movement planning, and show signatures of competitive value-based selection among these goals (Klaes, Schneegans, Schner, & Gail, 2012). In such a context, the development of sensorimotor behaviours has been investigated in robotic systems towards analysing behaviour formation, learning and imitation (Ugur, Nagai, Sahin, & Oztop, 2015). In a first stage, the robot is initialized with a basic movement capability, and discovers a set of behaviour primitives by exploring its movement parameter space. In the next stage, the robot exercises the discovered behaviours on different objects, and learns the caused effects; effectively building a library of affordances (Gibson et al., 1966) and associated conditions (i.e. predictors). Finally, in the third stage, the learned structures are used for more complex environment interactions. While most sensorimotor behaviours are manually engineered, there are attempts to learn more sophisticated, complex behaviours from simpler basic movements (Conradt, Galluppi, & Stewart, 2015; Stewart, Mundy,

Kleinhans, & Conradt, 2016). These basic maneuvers are still manually engineered relating sensory cues to simple movements like driving forward with no obstacle in the sensors field of view, turning with an obstacle in front of the robot or driving backwards when being close to an obstacle. In Conradt et al. (2015) the authors describe a method of learning more sophisticated behaviours from recorded sensorimotor data obtained from driving the robot by remote control as training examples, which can be considered as a supervised learning approach. In Stewart et al. (2016) the training examples are taken from recording data of the robot driving around without human interference and just labeling those situations as positive examples when the robot performed the desired action by accident, which can be considered as reinforcement learning. Both approaches are implemented on a small robot with the DVS (Dynamic Vision Sensor) (Lichtsteiner, Posch, & Delbruck, 2008) as sensory input using Nengo (Neural Engineering Objects) (Bekolay et al., 2014) and the NEF (Neural Engineering Framework) (Eliasmith & Anderson, 2003) as well as its interface for running neural networks models on the SpiNNaker (Spiking Neural Network Architecture) hardware (Furber, Galluppi, Temple, & Plana, 2014). Such models inherently support learning through the neural processing substrate offered by NEF. In a similar work (Andry, Gaussier, Nadel, & Hirsbrunner, 2004) used a model of a visuo-motor map able to represent the arm end point's position in an egocentred space (constrained by the vision) according to motor information (the proprioception). Sensorimotor behaviours such as tracking, pointing, and sequence learning were then obtained as the consequence of different internal dynamics computed on a neural network triggered by the visuomotor map. This research landscape focuses on learning patterns either in the perceptual space of the agent or in its action space. Such a system allows the agent to autonomously build a set of discrete steps for explaining its interactions. The system is incremental in the sense that it does not need to keep a record of the complete interaction with the environment. This can be achieved by focusing the search for recurring patterns around change points in the sensorimotor signal (Mohammad et al., 2013). In such a context, policy search methods allow robots to learn control policies for a wide range of tasks, but practical applications of policy search often require hand-engineered components for perception, state estimation, and low-level control. Training the perception and control systems jointly end-to-end (Levine, Finn, Darrell, & Abbeel, 2016) developed a method that can be used to learn policies that map raw image observations directly to torques at the robot's motors in a real-world manipulation task. The policies were represented by deep convolutional neural networks trained with supervision provided by a simple trajectory-centric reinforcement learning method.

We consider our work as part of the neuromorphic engineering and cognitive modelling research landscape. In this context, researchers developed computational models of particular brain parts (Oess, Krichmar, & Röhrbein, 2017) or drew inspiration from rats' cognitive capabilities (Barrera & Weitzenfeld, 2008) to tackle different problems in robot navigation. Others combine biologically inspired algorithms with neuromorphic hardware in closed-loop robotics systems using e.g. the principle of the NEF (Galluppi et al., 2014) or deep learning (Hwu, Isbell, Oros, & Krichmar, 2017). We consider our work somewhat in the middle: we use biologically inspired principles to implement a specific task in simulated neurons using the NEF on real robot hardware. Although our approach allows future deployment on neuromorphic hardware, a closed-loop neuromorphic system is not our focus in this work. However, to our knowledge, this paper is the first work to implement mobile robot manipulation completely in Spiking Neural Networks and to demonstrate it on real robot hardware.

2. Materials and methods

2.1. Hardware setup: OmniArmBot

The mobile manipulator used in this project is comprised of a custom developed omni-directional mobile platform, depicted in Fig. 1, with embedded low-level motor control and elementary sensory acquisition. The on-board ARM7 micro-controller receives desired motion commands and continuously adapts three PID motor control signals to achieve desired velocities. The robot's integrated sensors include wheel encoders for estimating odometry, a 9 degrees of freedom IMU (Inertial Measurement Unit), a bump-sensor ring which triggers binary contact switches, upon contact with objects in the environment and three silicon retinas providing visual sensor input. Two of these cameras are fixed on the mobile base, while one retina is attached to the end-effector to monitor the workspace of the robotic arm. The silicon retinas are eDVS (Lichtsteiner et al., 2008) and provide discrete events as response to temporal contrast. All 128×128 pixels of the DVS operate asynchronously and illumination changes are signalled within a few microseconds after occurrence (without having to wait for a frame to send information). Such information is communicated through spikes, representing a quantized change of log intensity at a particular location. The mobile platform is equipped with a 6-axis robotic arm with a working space between 10 cm and 41 cm. The robotic arm is composed of a set of links connected together by revolute joints and has a lifting weight force up to 800 g. The mobile platform contains an on-board battery of 12 V @ 30 Ah; thereby providing a total of 360 W, which allows autonomous operation for well above 5 h.

2.2. Software setup

The overall software architecture is based on a modular design allowing the extension of the current sensorimotor

capabilities of the robotic platform by adding more sensors and actuators. The software architecture is comprised of an embedded sensorimotor platform running on-board the robot and a neurocomputing platform suitable to run on various computing backends like CPU (Central Processing Unit), GPU (Graphics Processing Unit) and NPU (Neuromorphic Processing Unit), as shown in Fig. 2. The embedded platform is responsible for the low-level sensory perception, low-level motor control, and the bi-directional communication (i.e. outgoing data streaming and incoming commands) with the neurocomputing platform. Decoupling low-level sensorimotor control from the high level behaviour, the neurocomputing platform offers a generic interface to implement neurocontrol algorithms. This is achieved by separating the representation of the sensorimotor streams, the transformation to be applied to these streams and the actual dynamics of the algorithm. Using such a decoupling and a high-level description of the task, the neurocomputing platform acts as a neural compiler. Such a neural compiler is able to encode real-world sensorimotor streams in spiking activity over populations of neurons. This representation is able to support efficient computation, when deployed on dedicated hardware, as well as learning needed in closed-loop robotic applications, where data uncertainty, noise and unstructured environments require adaptive behaviour. Supporting intrinsically parallelizable processing mechanisms (i.e. neural networks), the neurocomputing platform can accelerate computation by natively mapping the neural controller on parallel computing hardware.

2.2.1. NEF/Nengo enabled neurocontrol

Nengo (Bekolay et al., 2014) is a Python library for building and simulating large-scale brain models using the methods of the NEF (Eliasmith & Anderson, 2003). Nengo can create sophisticated neural simulations for real-time closed loop robotic systems and is at the core of our neurocomputing platform. Nengo acts as a neural compiler using three main principles: representation, transformation and dynamics. For the first principle, representation, Nengo uses a distributed, non-linear encoding of continuous signals to represent them in the spiking activity over populations of neurons. Second, the neural populations encoding such values can have connections weights that allow them to approximate an arbitrary function of the inputs, which is referred to as the transformation principle. Finally, recurrent connections among the encoding populations allow the definition of complex dynamical models - the dynamics principle. Further details on the principles and workings of the NEF and its implementation in Nengo can be found in Eliasmith and Anderson (2003), Eliasmith et al. (2012) and Bekolay et al. (2014).

Our overall goal is to develop robotic control systems that are programmable and, at the same time, implemented as neural networks. We want them to be programmable so that we can leverage expert knowledge about the steps required to perform a task. We want them to be implemented as neural networks partly so that we can make use of energy-efficient neuromorphic hardware, but mostly because neural networks allow for gradual improvement of task performance via learning. However, neural networks

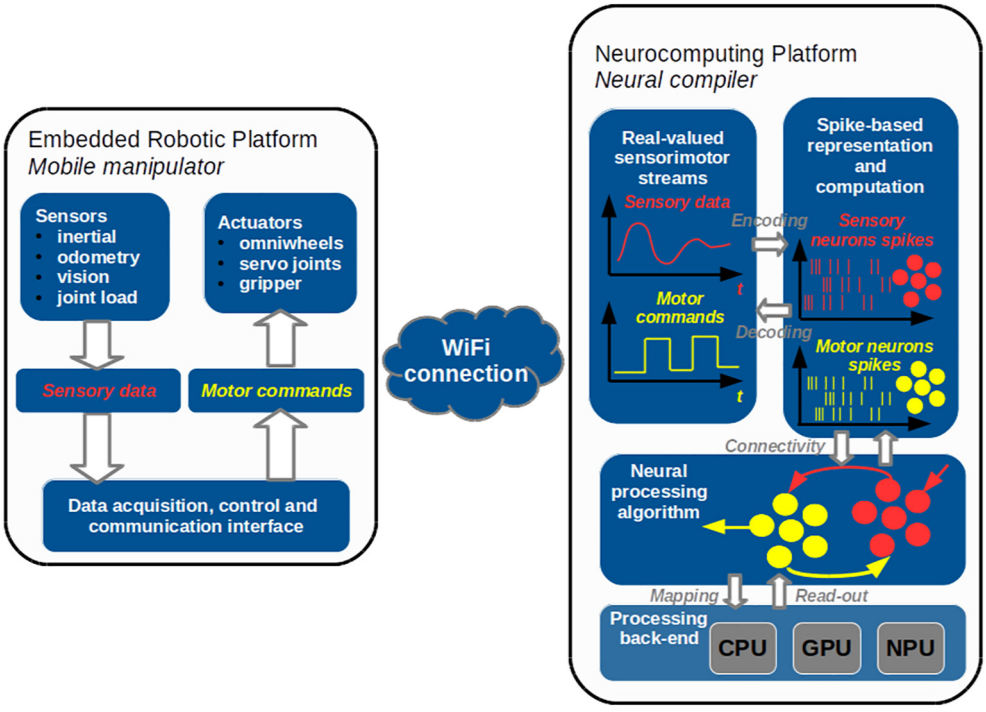

Fig. 2. Generic architecture: embedded robotic platform and neurocomputing platform.

on their own generally start with zero knowledge (random connection weights) and can often take a long time to learn, or completely fail to learn complex tasks. What we thus need is a method for taking a complex algorithm (i.e. the program we want to implement) and breaking it down into smaller components. Each of those components can then be implemented in a neural network. These individual neural networks can then be combined into one large neural network that can perform the entire task. This final network can then be implemented in neuromorphic hardware, and it could also be used as the starting point for further learning. Crucially, it should be noted that we are not trying to implement a perfect version of a task. Rather, we want to program a somewhat competent, initial version of a task that could then be further refined using a variety of neural network learning algorithms. In particular Duan, Chen, Houthooft, Schulman, and Abbeel (2016), provide a comprehensive survey of Deep Reinforcement Learning algorithms that are suitable for robotic control learning. Most of these algorithms are based around adjusting the connection weights of a neural network in order to improve performance on a task, based on occasional positive and negative reward values. Our long-term research goal is to apply these learning systems to the programmed neural networks that we describe in this paper.

2.2.2. Building algorithms with neural networks and Nengo

First, we start with the principle that a neural network is a function approximator. That is, a neural network takes in a real-valued input x (which may be multidimensional) and generates a real-valued output y (which may also be multidimensional). When we train a neural network, we generally provide it with a set of input-output pairs (i.e. a set of input values x_i and a corresponding set of output values y_i). This can be thought of as an implicit description of a function f, such that we desire $y = f(x)$. This standard depiction of a neural network is shown in Fig. 3a, with a 3-dimensional input x and a 2-dimensional output y. In order to build larger systems with more complex algorithms, we can combine multiple networks together. For example, Fig. 4a shows a system where we not only have $y = f(x)$, but we also have $z = g(w)$, and our final output is a function of the outputs of both of the other two networks, $m = h(y, z)$. By creating networks to compute these intermediate functions and then combining them together, we can implement complex computations. One crucial question, however, is what advantage do we get by breaking this complex function down into smaller, simpler functions? After all, it would have been possible to just make one single network that directly approximates the desired function, as shown in Fig. 4b. The primary reason not to combine functions together, as in Fig. 4b, is that of scaling. As algorithms become more complex, it becomes harder and harder for neural networks to approximate them. The traditional solutions are to either increase the number of neurons in the middle layer, or to add more layers. Both of these approaches make it harder for the learning algorithms to find the connection weights that best approximate the function, and make the network require more neurons, and thus more computational resources are needed. Importantly, one can think of Fig. 4a as the result of starting with Fig. 4b, adding in more hidden layers, and adjusting the connectivity. In other words, by building a complex network out of smaller networks, we are imposing structure on the network. We are specifically indicating that y and z are useful intermediate results that the network should find as an intermediate step before computing m. If we, as programmers, are correct in our decisions about this intermediate structure, then we can greatly simplify the process of creating these networks.

This approach to building large systems of neural networks out of smaller networks is the basis of the NEF (Eliasmith & Anderson, 2003) and the software toolkit Nengo (Bekolay et al., 2014). It has been applied to a wide variety of tasks and works for both spiking and non-spiking neuron models. Importantly, its ability to scale to large

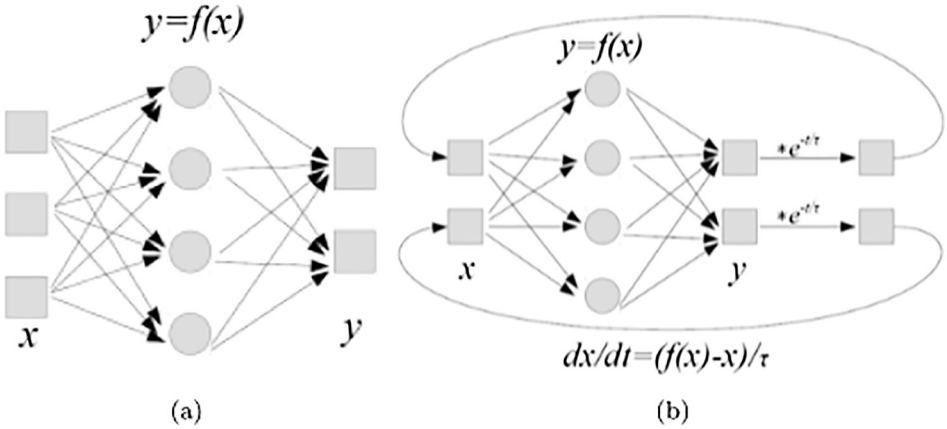

Fig. 3. (a) A simple neural network with a 3-dimensional input x and a 2-dimensional output y. By adjusting the connection weights between the layers, the network can approximate a desired function $y = f(x)$. (b) shows a similar network approximating a differential equation by using recurrent connections.

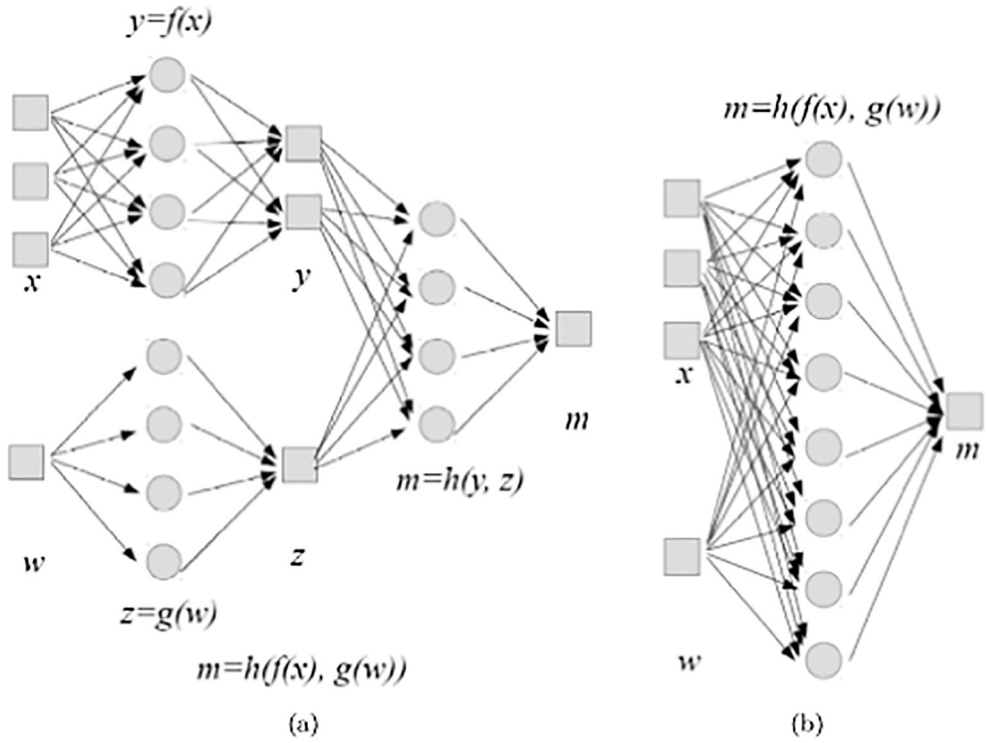

Fig. 4. Two neural networks computing a complex function $m = h(f(x), g(w))$. (a) shows a combined network computing $m = h(f(x), g(w))$ while the network in (b) computes the function directly.

complex systems allowed it to be the basis of the creation of Spaun, the first large-scale brain simulation capable of performing multiple behavioural tasks (Eliasmith et al., 2012). Finally, it should be noted that Nengo and the NEF also support recurrent connections. In particular, (Eliasmith & Anderson, 2003) show that by adding in a model of the post-synaptic filter (a low-pass filter observed in biological systems and implemented in most neuromorphic hardware), we can use a recurrent connection to cause the neural network to approximate a differential equation. As shown in Fig. 3b, if we create a network that approximates $y = f(x)$ and connect its output back to itself via a low-pass filter with a time constant τ, then we will generate a recurrent neural network that approximates the differential equation $\partial x/\partial t = \frac{f(x)-x}{\tau}$. Given these tools, we can use Nengo to construct large neural networks by combining and linking smaller ones. In order to do this, we have to take our desired algorithm and break it into small parts to be implemented in sub-networks, each of which is a function computed on an input vector or a differential equation.

2.2.3. Interface infrastructure

The interface between the embedded platform and the neurocomputing platform is designed to abstract the elementary data acquisition and control. It encapsulates them in spiking neural activity of neuron populations that represent sensory data or generate motor commands as provided by Nengo. Such an interface allows the neurocomputing platform to natively operate on either real-valued encodings of the sensory data and motor commands or their spike based representations.

3. Neural algorithm implementation in Nengo

Even the simplest behaviours are exploiting relations (i.e. functions) between sensory streams and motor commands. In order to design a neural controller able to adaptively switch control policies it is essential to extract such functions in a reliable way, given the variability and uncertainty in the sensorimotor streams. The relatively complex and nonlinear interactions among the sensors on the robot and its actuators, make the derivation of analytical forms of such functions hard. Alleviating the need for such precise and task-dependent modelling, neural networks are able to approximate such functions just from observations of the available sensorimotor streams. In general, adaptive behaviour is regarded as autonomous when the actions performed by the agent result from the interaction between its internal dynamics and the environment. Following such a perspective, our system is able to incrementally build up complex behaviours by superimposing more simple, basic behaviours.

In the current instantiation of our framework, we want to solve a non-trivial mobile manipulation task. Using our mobile platform the task is to manipulate objects with LED stimuli blinking at different frequencies to bring them in order. More precisely, the task can be seen as a grasp and sort task, in which the robot will select a certain con-

trol policy depending on the current sensory streams (mainly visual input) to find the misplaced stimulus and place it in the correct location. The main sensory input to our system is visual information detected by two eDVS mounted on top of the base and one eDVS attached to the end-effector of the robot's arm. We use an existing algorithm (Müller et al., 2011), which runs on-board of the eDVS's micro-controller, to track several LED-markers blinking at different frequencies. Therefore, we only need to send the output data of the tracking algorithm, namely x, y-position, radius and certainty for each tracked stimulus, via WiFi to the neurocomputing platform instead of three 128×128 event-streams. The tracking result (position and radius) is obtained by counting DVS-events over a certain time-window (see Müller et al., 2011 for details), while the percentage of events within the track-radius serves as a measure of certainty. Action selection is performed using a model of a biologically plausible neural circuit, namely the Basal Ganglia. The Basal Ganglia, according to Stewart et al. (2010), is an action selector that chooses whatever action has the best "salience" or "goodness". Selection is done on the basis of a context dependent utility signal for each possible action. Actions that are inappropriate for the current context may have low utility, and the task of the Basal Ganglia is to select the action that currently has the highest utility value. The Basal Ganglia will choose between the four possible behaviour stacks: Grab, Hold And Move Side, Put Down or Finish. We design our control network by defining a set of intermediate-level networks, namely Grab (see 3.2.2), Hold And Move Side (see 3.2.3), Put Down (see 3.2.4) and Finish (see 3.2.5) that are composed of various different low-level behaviours, and then we create a high-level network (see 3.2.6) whose outputs activate and deactivate the low-level networks. To accomplish this, each of the low-level networks will have an input a which indicates its level of activation. If a is 0 then the output from that network should be 0, and if a is 1 then it should perform its basic activity. It should be noted that this sort of control system design is strongly reminiscent of the classic subsumption architecture (Brooks, 1986). All behaviours are implemented in the software suite Nengo, which is used to translate our functional descriptions to spiking neuron representations and to combine all sub-task networks to one large network able to solve the whole task.

3.1. Perception and motor-control: low-level "reflex" behaviours

3.1.1. Orient left/right using all cameras

This behaviour uses the x-position location of the target in all three camera views to control the rotation of the robot base. If the object is on the left, it turns left, and if it is on the right, it turns right. This should cause the robot to turn to face the object. If it cannot see the object, it does nothing. If the object cannot be seen by a particular camera, it does not contribute.

3.1.2. Orient left/right using arm camera only

This behaviour is similar to 3.1.1 in the sense that it controls the rotation of the robot base. Here, instead of using both base cameras, we use the arm retina to detect the target. This is meant to be used when the arm is already in grasping position (cf. behaviour 3.1.4 and Fig. 14b) so that the arm camera has a good view of the target, allowing for a more fine-grained close-up control than behaviour 3.1.1 to move the gripper to the correct place for grasping the target (cf. Fig. 14c).

3.1.3. Move forward/backward to grasping distance

Here, we use an approximation of binocular disparity (i.e. the difference between x-values in the image of the left and right base retina) to estimate the distance of the robot base to the object in order to control whether we should move forward or backward. This behaviour will output 0 if the object is not mostly in front of the robot (as computed by averaging the x positions in the left and right camera). If it is in front of the robot, then we move forward or backward to achieve the desired distance, which is an adjustable parameter.

3.1.4. Move arm to grasping position

This behaviour moves the arm from its resting position to a position suitable for grasping (Fig. 14b).

3.1.5. Move backwards

This behaviour causes the robot base to move backwards.

3.1.6. Close grip

This simply closes the gripper.

3.1.7. Move sidewards

This behaviour moves the robot base sidewards towards a goal position and rotates the base to keep the goal position in the middle of the field of view of the robot. We make use of the target positions of the neighbouring objects (cf. Out of Order Network 3.2.1), where the middle between them is the goal position of this behaviour.

3.1.8. Move arm to put-down position

This behaviour simply moves the arm from its holding position to a position suitable for putting down the object again.

3.2. Reasoning: high-level "cognitive" behaviours

3.2.1. Out of order network

This network computes the object to manipulate and serves as basis for all following behaviours built on top of it. Fig. 5 gives a schematic visualization of the network and its individual components. Assuming the objects' blinking frequencies are given in descending order, this network detects if one frequency does not fit the assumed order by calculating the pairwise difference between x-

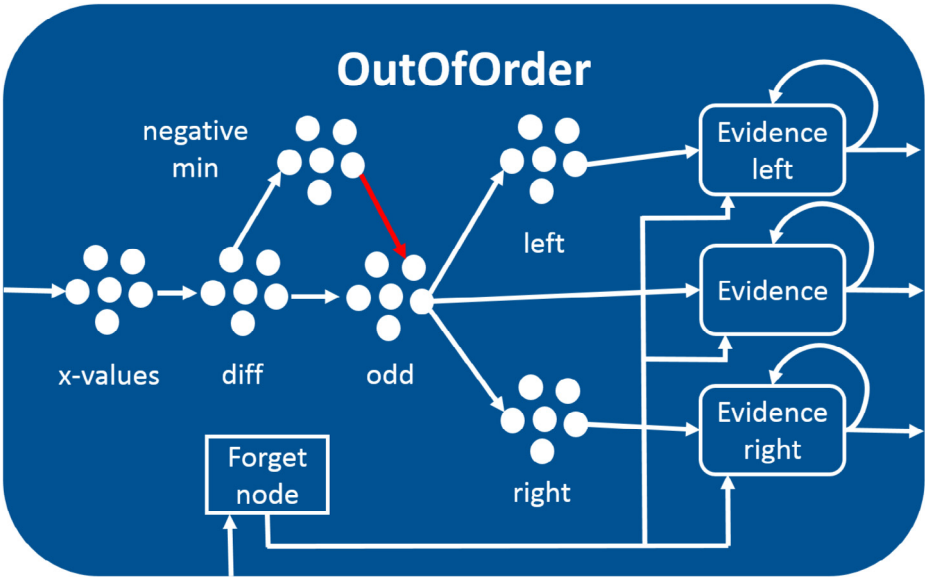

Fig. 5. Schematic visualization of the Out of Order network 3.2.1, which detects if one frequency does not fit the assumed order as target and keeps this objects information in memory. Sets of white circles indicate neural populations while boxes indicate sub-networks. The *x*-values ensemble encodes *x*-positions of stimuli in DVS-image, the diff ensemble encodes pairwise differences between *x*-positions, the negative-min ensemble indicates if the minimal difference is negative, odd encodes the frequency, which is out of order (inhibited by negative min when all differences are positive), the evidence networks integrate evidence for the target object and the neighbour frequencies if in correct order.

positions in the DVS image. The objects are in order, if and only if all pairwise differences are non-negative (in this case, the negative min ensemble inhibits the odd ensemble, visualized by a red arrow in Fig. 5), while otherwise at least one object is out of place (cf. 5 neural ensembles *x*-values and diff). The corresponding object is indicated as target and its information is stored in memory (cf. Fig. 5 neural ensemble odd and sub-network evidence). Furthermore, the network detects those frequencies, which should be the neighbours of our detected target if in correct order and keeps their information in memory as well (cf. Fig. 5 neural ensembles left/right and sub-networks evidence left/right). Memory in the evidence sub-networks is realized as dynamical system (cf. principle dynamics in Section 2.2.1 or (Bekolay et al., 2014; Eliasmith & Anderson, 2003)) with recurrent connections. Thereby, the networks integrate evidence and keep the desired information available even in the absence of sensory input until a forget mechanism is triggered (cf. forget-node in Fig. 5). This mechanism can be triggered manually by the user or automatically by another network.

Fig. 6 shows an example of actual DVS input data from the embedded tracking algorithm as well as the decoded output of the networks subcomponents activity: during the first 5 s the stimuli are in correct descending order from left to right in the DVS-image (Fig. 6a), so the minimum pairwise difference is non-negative (Fig. 6b and c). In the interval 5–15 s, the 250 Hz stimulus is put between the 150 Hz and 200 Hz stimuli, so now the sequence is out of order and the 250 Hz frequency is detected by the odd ensemble (Fig. 6d) while the evidence networks integrate accordingly (Fig. 6f–h). Starting from around 15 s the 250 Hz and 350 Hz stimuli are interchanged and the network's outputs change accordingly (Fig. 6a–d). However, the evidence networks - as desired - still keep the information about the old target until a forget mechanism (Fig. 6e) is triggered in the interval 20–25 s allowing the evidence networks to recover for new input (Fig. 6f–h).

3.2.2. Perform grasping action

This is a high-level behaviour that uses the low-level behaviours described in Section 3.1 to find and grasp the desired object. As long as the object is not visible in both base cameras, the robot moves backwards (behaviour 3.1.5). As soon as the robot reliably detects the object in both base cameras, the base moves forward (and backward if necessary) using behaviour 3.1.3 and moves the arm in the correct position for grasping (behaviour 3.1.4). If the robot can see the object with the arm camera, then it uses the arm camera for orientation (behaviour 3.1.2), otherwise it uses all three (behaviour 3.1.1). Also, if the robot can't see the object with the arm camera, then it backs up (behaviour 3.1.5). Note that behaviour 3.1.5 and behaviour 3.1.3 both move the robot forward and backward, so when they are both active the robot will end up achieving a position farther away from the object than when just behaviour 3.1.3 is active (the motor commands are summed). This positions the robot such that it is at the right distance for grasping. Finally, once the robot is at the correct distance from the object and right in front of it, then it closes the gripper. This combination of actions serves to successfully grab the object. Fig. 7 gives a schematic visualization of this network while Fig. 8 shows the activation levels of the high- and low-level behaviours (Fig. 8a and b) as well

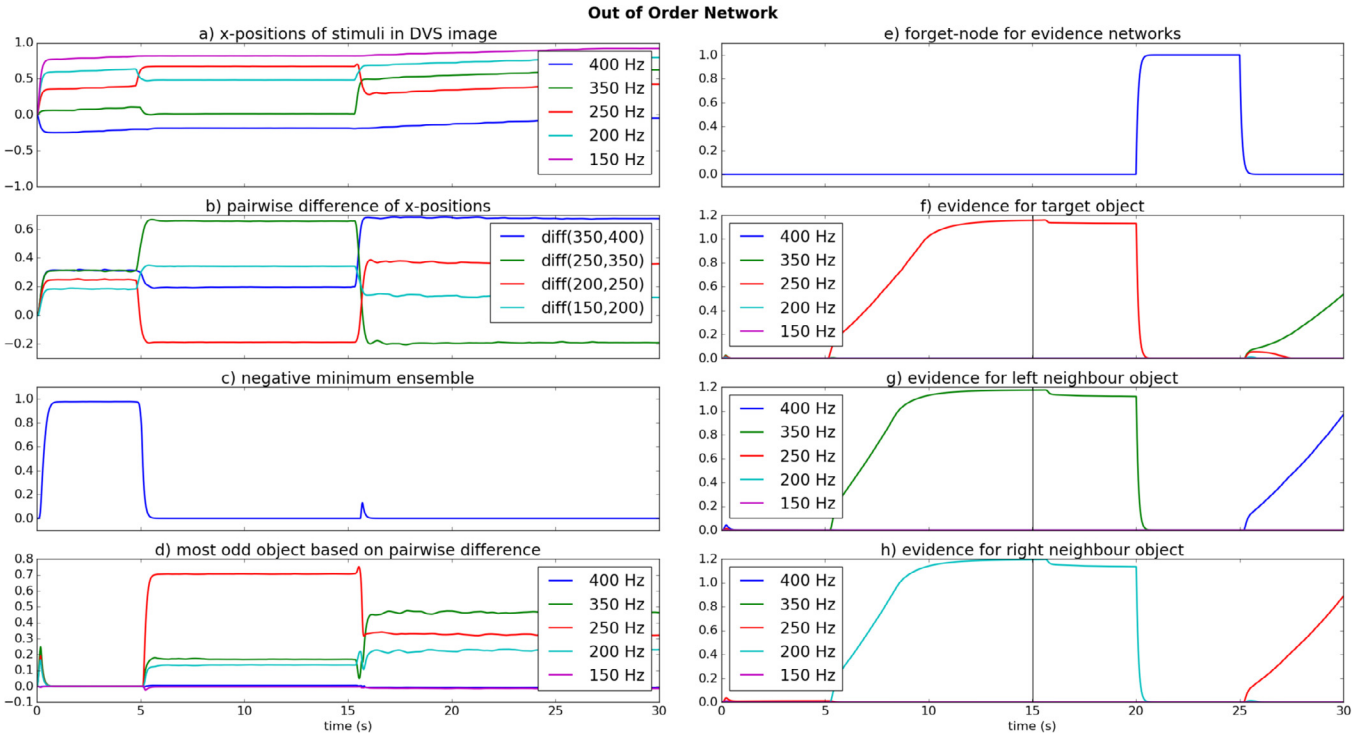

Fig. 6. Decoded output of the Out of Order network's 3.2.1 neural components based on DVS input data from embedded tracking.

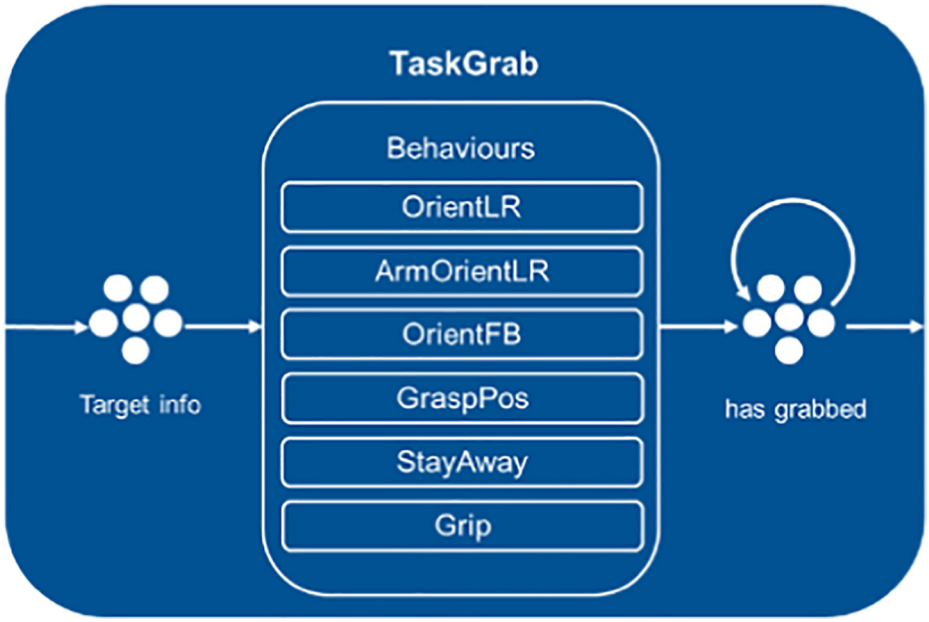

Fig. 7. Schematic visualization of the TaskGrab network 3.2.2, which finds an object and grabs it. Sets of white circles indicate neural populations while boxes indicate sub-networks. The TargetInfo ensemble encodes sensory information (x- and y-position in the image, as well as radius and track-certainty for each DVS) of the target object (coming from the evidence sub-network in the out of order network), which serves as input for the low-level behaviours. For detailed descriptions of those, see Section 3.1. The has-grabbed ensemble keeps the information once the object was grabbed in memory to indicate this task finished successfully.

as the sensory information the behaviours make use of, namely tracking certainty and disparity and x-position of the target object in the arm retina (Fig. 8c and d respectively).

3.2.3. Hold object and move to goal position

This behaviour combines holding an object by keeping the gripper closed while moving to the goal position at the same time. Here, we make use of the stored information

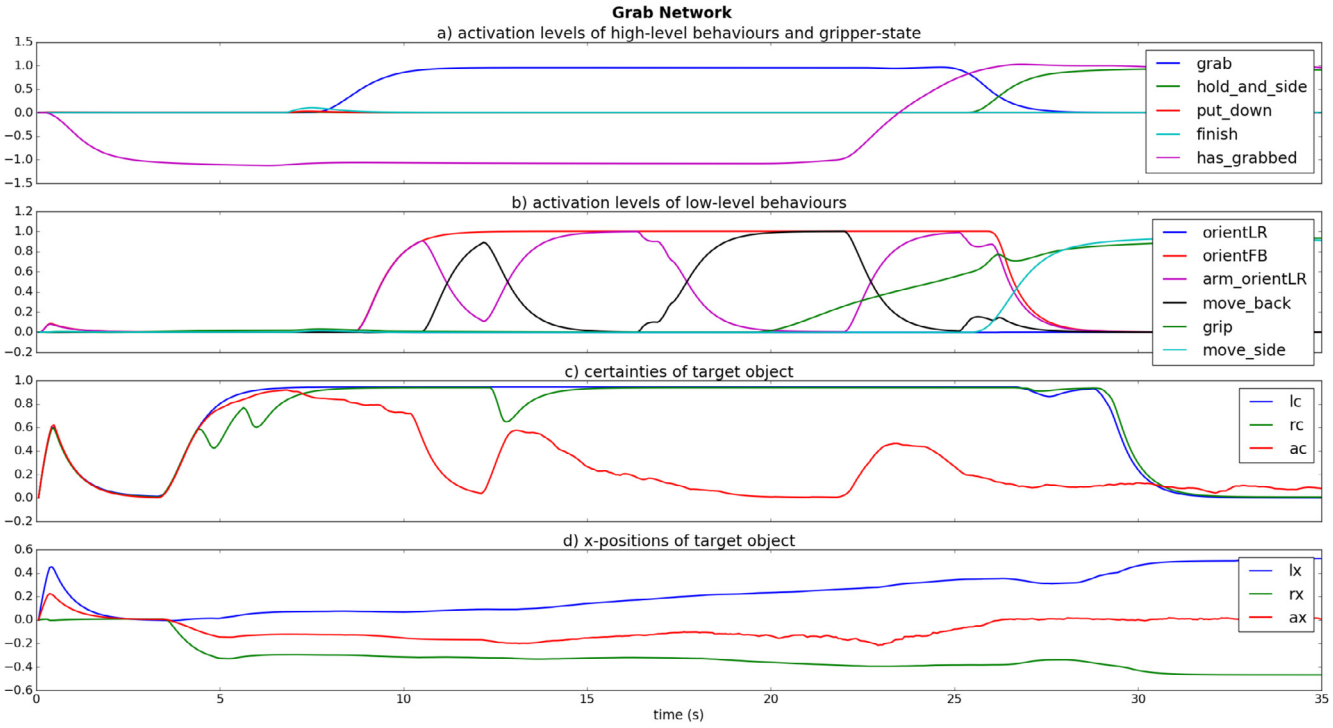

Fig. 8. Decoded output of the Grab network's 3.2.2 neural components.

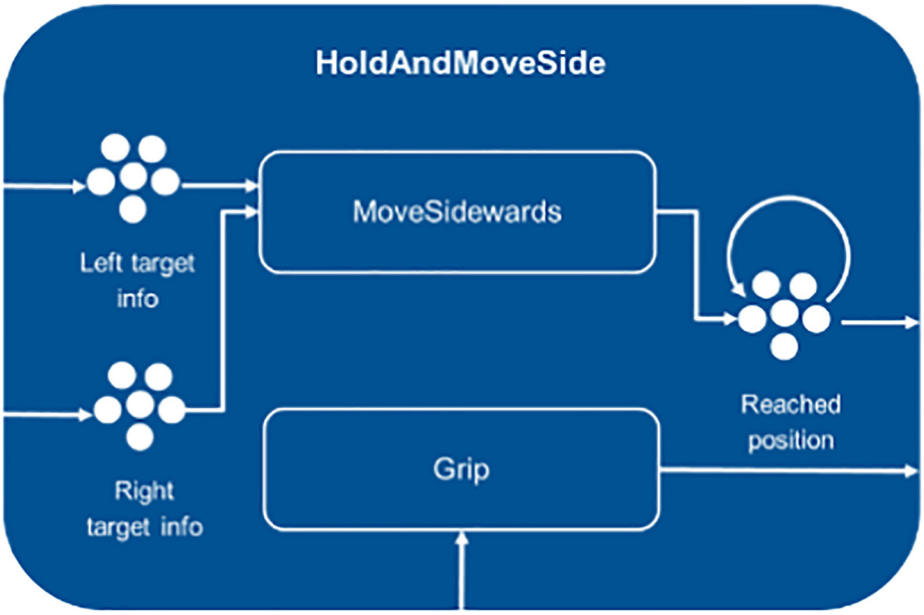

Fig. 9. Schematic visualization of the HoldAndMoveSide network 3.2.3, which holds the object while approaching the goal position for putting down the target object. Sets of white circles indicate neural populations while boxes indicate sub-networks. The Left/Right TargetInfo ensembles encode sensory information (x- and y-position in the image, as well as radius and track-certainty for each DVS) of the neighbour objects (coming from the left/right evidence networks in the out of order network), which serves as input for the low-level behaviours. The MoveSidewards behaviour 3.1.7 uses the mean value of the lateral positions of the left resp. right neighbour in the left resp. right base camera as an estimation of the middle between the neighbour objects and moves the base to this position, while the Grip behaviour 3.1.6 keeps the gripper closed. The Reached position ensemble serves as a memory integrating evidence once the goal position is reached to indicate that this subtasks finished successfully.

about the target object's neighbours to calculate the goal position. Therefore, we use the mean value of the lateral positions of the left neighbour in the left base camera and the right neighbour in the right base camera as an estimation of the middle between the neighbour objects, which is where we want to place our target object. This value is

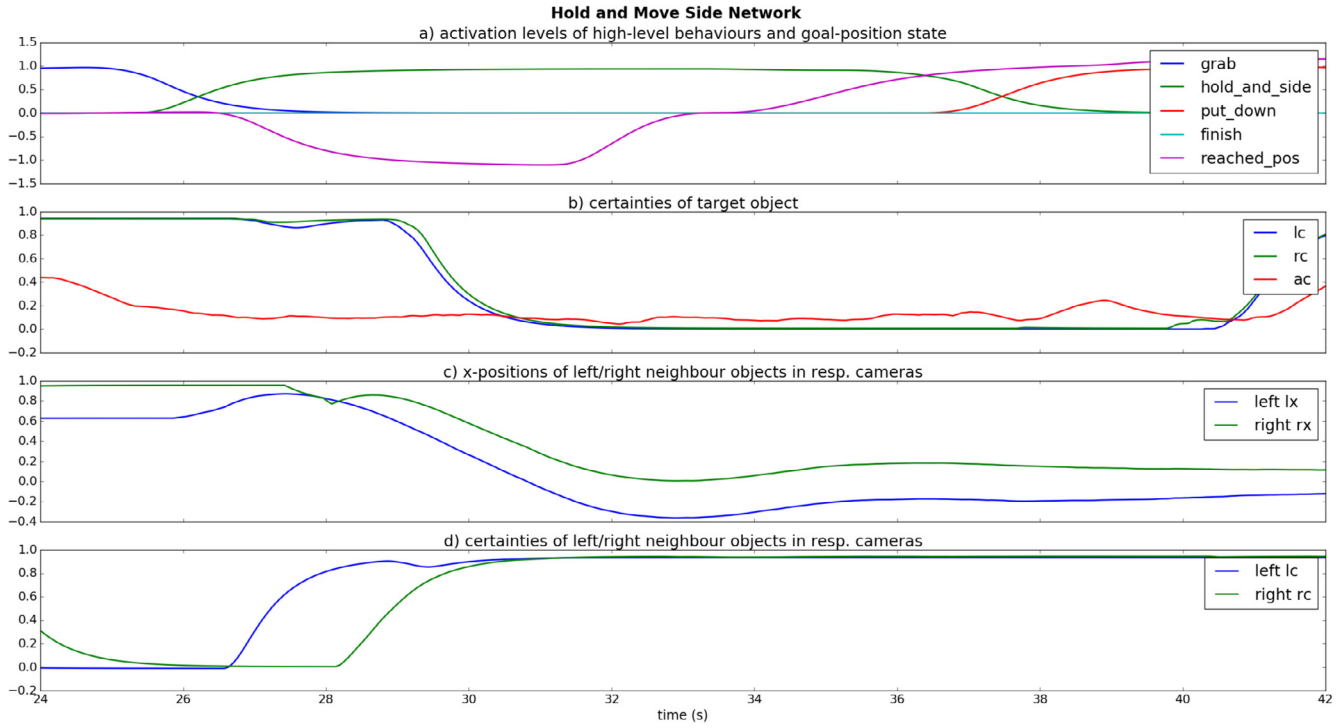

Fig. 10. Decoded output of the Hold and Move Side network's 3.2.3 neural components.

used to control sidewards and rotation motion of the base to navigate the robot to the correct position for putting down the target object (see Fig. 10c in the interval from 28 s to 37 s). Fig. 9 gives a schematic visualization of this network. Fig. 10 shows the activation levels of the high- and low-level behaviours.

3.2.4. Put object down

This behaviour simply moves the arm from its gripping position to a position suitable for releasing an object, while opening the gripper and moving the base slightly backwards at the same time to ensure smooth and safe placement of the target object.

3.2.5. Finish task

This behaviour simply makes the robot base back off from the manipulated objects. After stopping the base - implicitly by deactivating all other behaviours - the arm moves to back to resting position automatically, which indicates that the whole sequence of tasks is completed.

3.2.6. Perform sorting task

This is a high-level behaviour combining all the other behaviours described so far to complete the whole task of moving the target object to its correct position in the sequence of frequencies. Fig. 12 gives a schematic visualization of the network. To choose the appropriate action to take, we used models of the Basal Ganglia and Thalamus (Stewart et al., 2010), which are available as pre-implemented networks in Nengo. Corresponding to each of the behaviours described so far in Section 3.2, we created a neural ensemble encoding input values for the Basal Ganglia network to choose from (cf. choice ensemble in Fig. 12). This ensemble represents the current state of the sub-networks, namely the evidence value indicating each network's level of success (namely the *has grabbed* ensemble in Fig. 7, the *reached position* ensemble in Fig. 9 and the *finished* ensemble in Fig. 11). Depending on the most salient

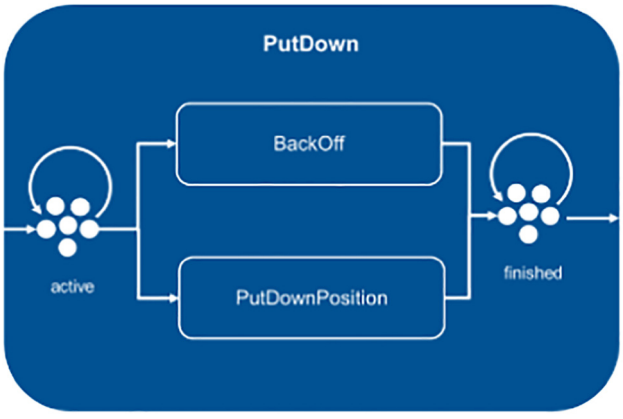

Fig. 11. Schematic visualization of the Put Object Down network 3.2.4, which moves the arm from holding position to put-down position, opens the gripper and moves the base slightly backwards to ensure smooth and safe placement of the target object. Sets of white circles indicate neural populations while boxes indicate sub-networks. The active ensemble integrates evidence when this behaviour is initially activated, the PutDownPosition behaviour 3.2.4 moves the arm to a position suitable for releasing an object while the BackOff network activates the low-level behaviour to move backwards 3.1.5. The finished ensemble integrates evidence once the PutDown task is completed to indicate that this subtask finished successfully.

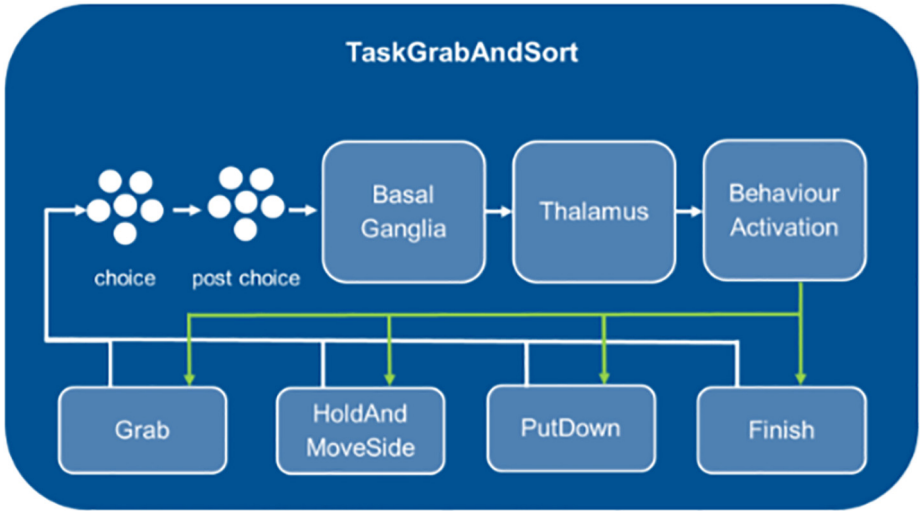

Fig. 12. Schematic visualization of the TaskGrabAndSort network 3.2.6. Sets of white circles indicate neural populations while boxes indicate sub-networks. The boxes in the lower part visualize the subtask-networks (3.2.2–3.2.5), which are activated by the upper network chain for action selection incorporating the Basal Ganglia and Thalamus networks pre-implemented in Nengo.

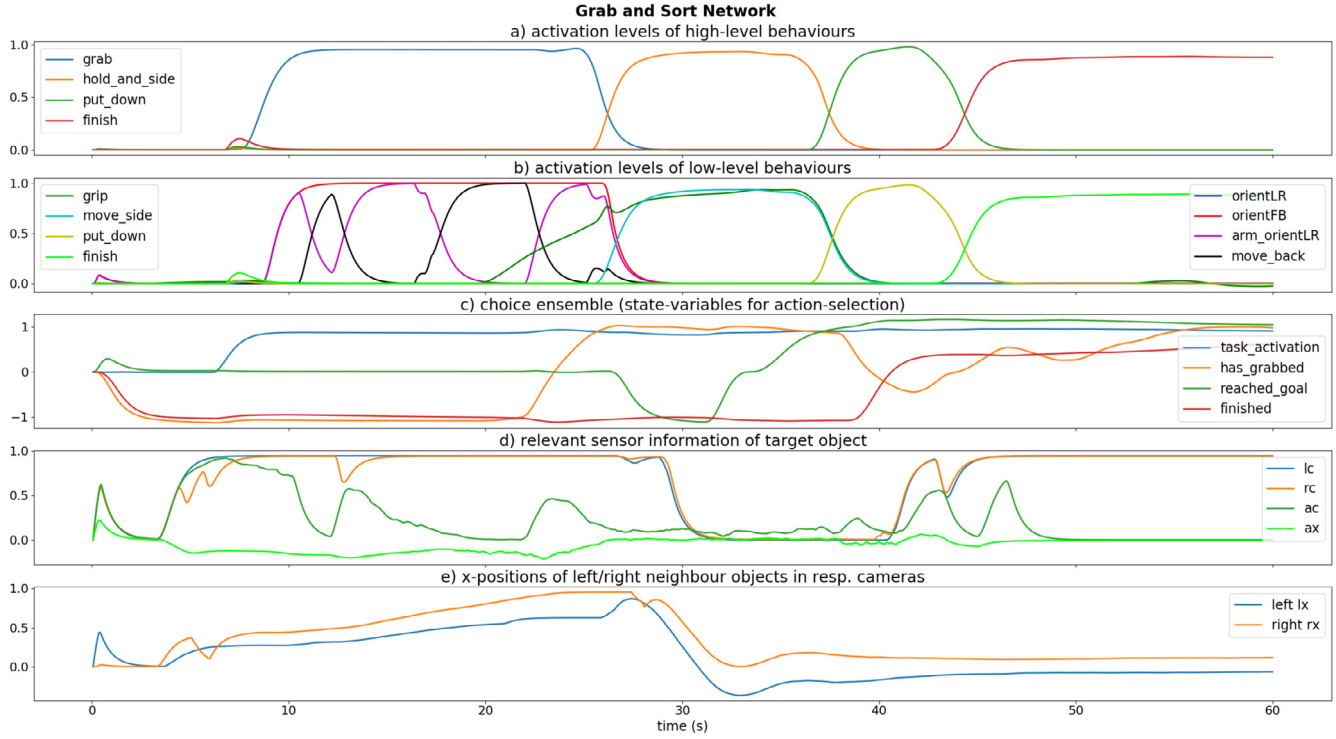

Fig. 13. Decoded output of the TaskGrabAndSort network's 3.2.6 neural components. This visualizes one example run of the complete task and shows the interplay between high-level and low-level behaviours, state-variables for action-selection and sensory-data.

signal in the choice ensemble, the Basal Ganglia network sends activation signals to the high-level behaviour sub-networks (visualized with green[1] arrows in Fig. 12). Initially, Perform Grasping Action (behaviour 3.2.2) is enabled. Once the robot picked up the target object and built up sufficient evidence, the Basal Ganglia network activates the behaviour to hold the (target) object and navigate the robot to the goal position (behaviour 3.2.3). As soon as the robot reached its goal position between the neighbour objects and the according network built sufficient evidence, the behaviour 3.2.4 to put down the target object is activated. As soon as this behaviour is completed, the whole sequence is wrapped up by activating the Finish behaviour 3.2.5.

Fig. 14 illustrates the most important stages of one example run while Fig. 13 gives a visualization of the

[1] For interpretation of color in Fig. 12, the reader is referred to the web version of this article.

decoded output of the networks components: once the Out of Order network detected the target object (roughly the first 5 s), the Grab behaviour is activated to find grasp the target object (Fig. 14a–c, Fig. 13, 8–26 s). The different low-level behaviours for orientation and navigation are enabled based on the certainty and disparity of the tracked stimuli (for details see Section 3.2 behaviour 3.2.2). Once the robot grabbed the target object, the Basal Ganglia activates the Hold and Move Side behaviour 3.2.3 (Fig. 14d and e, Fig. 13, 26–39 s). A clear indicator for successful pick-up is the decrease of certainty (cf. Section 3) in the base-retinas (cf. values of lc and rc around 30 s in Fig. 13d and Fig. 14d and e), which means that the target object is no longer visible in the base retinas. The main sensory input in this phase is the x-position of the neighbour objects (Fig. 13e) with the robot aiming for the middle between the two. Once the robot reached its goal position between the neighbouring objects, the Put Down behaviour 3.2.4 is activated (Fig. 14f and g). Finally the whole task is wrapped up by the Finish behaviour 3.2.5 and all other behaviours are deactivated (Fig. 14h) to put the robot back into resting position.

4. Discussion

We have shown a mobile robotic manipulator capable of solving a pick-and-place task, with algorithmic components completely implemented in the framework of Spiking Neural Networks. In this work, we focused on task performance only, i.e. our emphasis was on the generation of a network that is capable of performing the pick and sort task with a minimum of prior information and parametrization. We aimed for a proof-of-concept implementation showing general applicability of our approach: we intend to expand its functionality with supervised and unsupervised learning methods to improve task performance in future work. For this reason, we omitted a detailed analysis of precision regarding robot motion and grasping actions, which is not our main concern here. Similarly, we did not extensively investigate flexibility of our implementation in the sense,

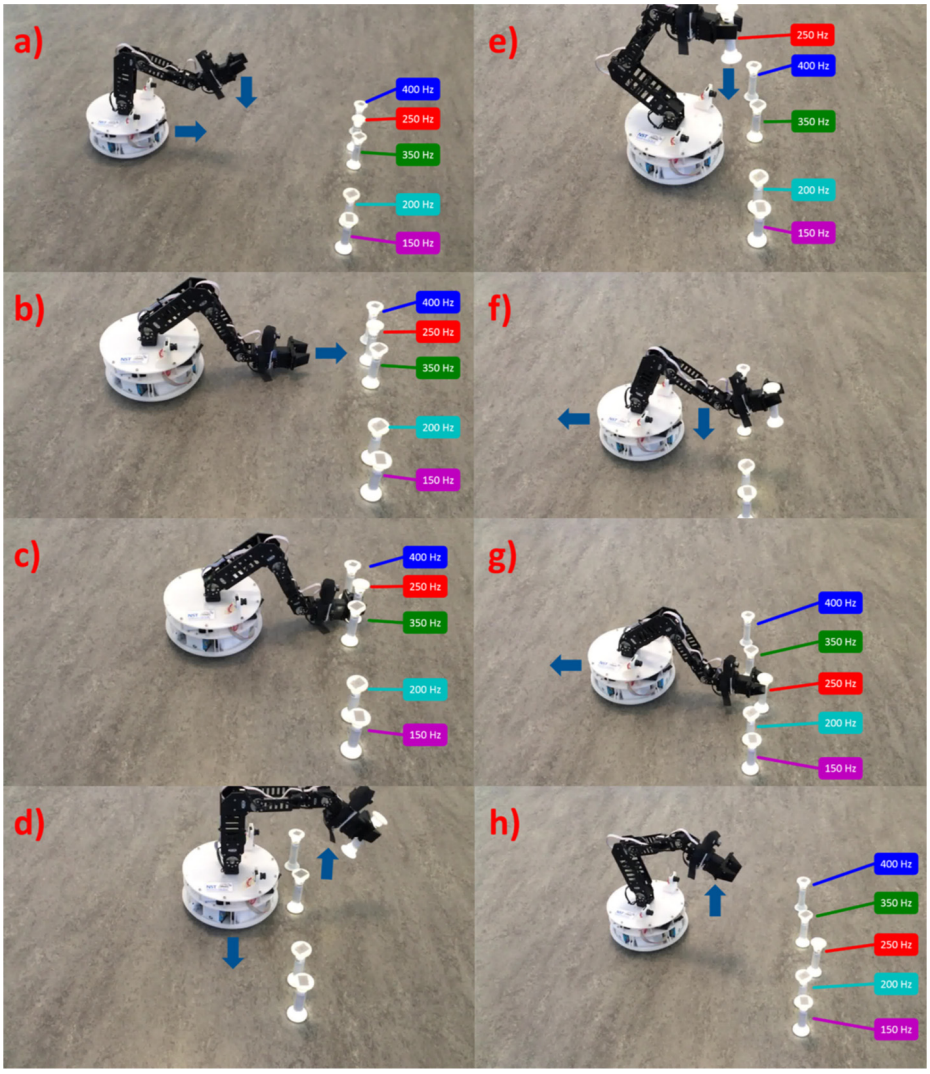

Fig. 14. Selected stages of an example run of the Grab and Sort Task 3.2.6.

that our approach is currently unable to compensate for faults on hardware level. However, the neural framework and modular software architecture are intrinsically flexible and allow extending the current implementation incrementally with additional network components, either manually designed or automatically learned.

Furthermore, the underlying Nengo-based neural compiler supports the use of dedicated neuromorphic hardware systems, which allows to run even large-scale neural networks in real-time. The possibility to introduce learning will eventually enable such systems to adapt their control policies and decision making to unpredictable changes in the environment. Furthermore, the neural implementation is beneficial in terms of allowing to combine networks hand-programmed by experts/engineers with learning networks that improve themselves over time with increasing data/experience. Here, our ultimate goal is to design systems that are capable of adapting to new task during operation time while at the same time using experience from previous tasks and expert knowledge.

4.1. Limitations

The current algorithmic implementation has some inherent limitations. For now, the network is only able to detect one object not fitting in the sequence of frequencies. To overcome this limitation, the Out of Order network 3.2.1 could be enhanced to solve the sorting-problem in a neural fashion. Another obvious limitation of the current implementation is the need for neighbour objects to find the goal position for put-down. This makes it impossible to solve the task for those objects blinking at a maximum or a minimum frequency. As for the first limitation, the Out of Order network needs to become more sophisticated to detect the goal position in these edge-cases when a border-object needs to be manipulated. One possibility could be to detect one neighbour and to use the maximum of the pairwise differences as distance estimation for the offset.

4.2. Future work

Our short-term goal is to enhance the current implementation and to solve the technical limitations mentioned in Section 4.1. A more intermediate topic for future work is to adapt the current system for test runs on other computing backends to benchmark performance and runtime. Nengo supports different simulation backends, for instance GPUs (Bekolay et al., 2014) and dedicated neuromorphic computing hardware like SpiNNaker (Furber et al., 2014; Mundy, Knight, Stewart, & Furber, 2015). Apart from these more technical issues, our long-term goal is to use the current implementation as a starting point for self-improving learning systems. As the current implementation is built in the framework of Spiking Neural Network it supports different learning approaches, e.g. supervised (MacNeil & Eliasmith, 2011), semi-supervised (Bekolay, Kolbeck, & Eliasmith, 2013) and reinforcement learning (Rasmussen & Eliasmith, 2014). One direction for future research could be to use the current low-level behaviours for initialization and to let the system learn and improve them incrementally by experience. For instance, one problem in learning robotic systems, e.g. policy search for motor control (Levine et al., 2016), is the acquisition of sufficiently large training data sets. Recording large amounts of training data by repeating one specific task with real robotic systems is time-consuming and often infeasible, while training-data from simulation is usually not realistic enough to capture the complexity of noisy real-world data. In these cases, a system which is able to use expert-knowledge for certain (sub-) tasks as a starting point could significantly speed-up the learning process. We believe that our current approach is a promising first step for further research in this direction.

Acknowledgement

This work was partly supported by the Bavarian Research Alliance, which awarded a BayIntAn Fellowship for fostering collaborations with CNRG, University of Waterloo.

Appendix A. Supplementary material

Supplementary data associated with this article can be found, in the online version, at https://doi.org/10.1016/j.cogsys.2018.03.006.

References

Andry, P., Gaussier, P., Nadel, J., & Hirsbrunner, B. (2004). Learning invariant sensorimotor behaviors: A developmental approach to imitation mechanisms. *Adaptive Behavior, 12*(2), 117–140. https://doi.org/10.1177/105971230401200203.

Barrera, A., & Weitzenfeld, A. (2008). Biologically-inspired robot spatial cognition based on rat neurophysiological studies. *Autonomous Robots, 25*(1), 147–169. https://doi.org/10.1007/s10514-007-9074-3.

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., ... Eliasmith, C. (2014). Nengo: A Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics, 7*(48). https://doi.org/10.3389/fninf.2013.00048<http://www.frontiersin.org/neuroinformatics/10.3389/fninf.2013.00048/abstract> .

Bekolay, T., Kolbeck, C., & Eliasmith, C. (2013). Simultaneous unsupervised and supervised learning of cognitive functions in biologically plausible spiking neural networks. In *35th Annual conference of the Cognitive Science Society* (pp. 169–174). Cognitive Science Society.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation, 2*(1), 14–23.

Butz, M. V., Herbort, O., & Hoffmann, J. (2007). Exploiting redundancy for flexible behavior: Unsupervised learning in a modular sensorimotor control architecture. *Psychological Review, 114*(4), 1015–1046.

Conradt, J., Galluppi, F., & Stewart, T. C. (2015). Trainable sensorimotor mapping in a neuromorphic robot. *Robotics and Autonomous Systems*. https://doi.org/10.1016/j.robot.2014.11.004.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19–24, 2016* (pp. 1329–1338). <http://jmlr.org/proceedings/papers/v48/duan16.html>.

Eliasmith, C., & Anderson, C. H. (2003). Neural engineering: Computation, representation, and dynamics in neurobiological systems. In *Computational neuroscience*. Cambridge, Mass: MIT Press.

Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., & Rasmussen, D. (2012). A large-scale model of the functioning brain. *Science, 338*(6111), 1202–1205. https://doi.org/10.1126/science.1225266<http://science.sciencemag.org/content/338/6111/1202>.

Furber, S., Galluppi, F., Temple, S., & Plana, L. (2014). The SpiNNaker project. *Proceedings of the IEEE, 102*(5), 652–665. https://doi.org/10.1109/JPROC.2014.2304638.

Galluppi, F., Denk, C., Meiner, M. C., Stewart, T.C., Plana, L.A., Eliasmith, ..., Conradt, J. (2014). Event-based neural computing on an autonomous mobile platform. In *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014* (pp. 2862–2867). doi:https://doi.org/10.1109/ICRA.2014.6907270.

Gibson, J. (1966). The senses considered as perceptual systems, Houghton Mifflin. <https://books.google.de/books?id=J9ROAAAAMAAJ>.

Hwu, T., Isbell, J., Oros, N., & Krichmar, J. (2017). A self-driving robot using deep convolutional neural networks on neuromorphic hardware. In *2017 International Joint Conference on Neural Networks (IJCNN)* (pp. 635–641). doi:https://doi.org/10.1109/IJCNN.2017.7965912.

Klaes, C., Schneegans, S., Schner, G., & Gail, A. (2012). Sensorimotor learning bi-ases choice behavior: A learning neural?eld model for decision making. *PLOS Computational Biology, 8*(11), 1–19. https://doi.org/10.1371/journal.pcbi.1002774.

Krakauer, J. W., & Mazzoni, P. (2011). Human sensorimotor learning: adaptation, skill, and beyond. *Current Opinion in Neurology, 21*(4), 636–644.

Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research, 17*(1), 1334–1373<http://dl.acm.org/citation.cfm?id=2946645.2946684>.

Lichtsteiner, P., Posch, C., & Delbruck, T. (2008). A 128x128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits, 43*(2), 566–576. https://doi.org/10.1109/JSSC.2007.914337.

MacNeil, D., & Eliasmith, C. (2011). Fine-tuning and the stability of recurrent neural networks. *PLOS ONE, 6*(9), 1–16. https://doi.org/10.1371/journal.pone.0022885.

Mohammad, Y. F. O., Nishida, T. (2013). Learning sensorimotor concepts without reinforcement. In *Lifelong machine learning, Papers from the 2013 AAAI spring symposium, Palo Alto, California, USA, March 25–27, 2013*. <http://www.aaai.org/ocs/index.php/SSS/SSS13/paper/view/5699>.

Müller, G. & Conradt, J. (2011). A miniature low-power sensor system for real time 2D visual tracking of LED markers. In *Proceedings of the IEEE international conference on robotics and biomimetics (IEEE-ROBIO)*.

Mundy, A., Knight, J., Stewart, T., & Furber, S. (2015). An efficient SpiNNaker implementation of the Neural Engineering Framework. In *2015 International Joint Conference on Neural Networks (IJCNN)* (pp. 1–8). doi:https://doi.org/10.1109/IJCNN.2015.7280390.

Oess, T., Krichmar, J. L., & Röhrbein, F. (2017). A computational model for spatial navigation based on reference frames in the hippocampus, retrosplenial cortex, and posterior parietal cortex. *Frontiers in Neurorobotics, 11*, 4. https://doi.org/10.3389/fnbot.2017.00004<https://www.frontiersin.org/article/10.3389/fnbot.2017.00004>.

Rasmussen, D., & Eliasmith, C. (2014). A neural model of hierarchical reinforcement learning. In P. Bello, M. Guarini, M. McShane, & B. Scassellati (Eds.), *Proceedings of the 36th annual conference of the Cognitive Science Society* (pp. 1252–1257). Austin: Cognitive Science Society<https://mindmodeling.org/cogsci2014/papers/221/index.html>.

Stewart, T. C., Choo, X., & Eliasmith, C. (2010). Dynamic behaviour of a spiking model of action selection in the Basal Ganglia. In *10th International conference on cognitive modeling*.

Stewart, T. C., Mundy, A., Kleinhans, A., & Conradt, J. (2016). Serendipitous offline learning in a neuromorphic robot. *Frontiers in Neurorobotics, 10*(1). https://doi.org/10.3389/fnbot.2016.00001 <http://www.frontiersin.org/neurorobotics/10.3389/fnbot.2016.00001/abstract>.

Ugur, E., Nagai, Y., Sahin, E., & Oztop, E. (2015). Staged development of robot skills: Behavior formation, affordance learning and imitation with motionese. *IEEE Transactions on Autonomous Mental Development, 7*(2), 119–139. https://doi.org/10.1109/TAMD.2015.2426192.