Serendipitous Offline Learning in a Neuromorphic Robot

Terrence C. Stewart*, Ashley Kleinhans†, Andrew Mundy‡ and Jörg Conradt§
*Centre for Theoretical Neuroscience, University of Waterloo, Waterloo, ON, Canada
tcstewar@uwaterloo.ca
†Council of Scientific and Industrial Research, Pretoria, South Africa
akleinhans@csir.co.za
‡School of Computer Science, University of Manchester, Manchester, UK
andrew.mundy@ieee.org
§Neuroscientific System Theory, Department of Electrical and Computer Engineering, Technische Universität München, Germany
conradt@tum.de

***Abstract*—We demonstrate a neuromorphic learning paradigm that is well-suited to embodied learning of complex sensorimotor mappings. A mobile robot is first controlled by a basic set of reflexive hand-designed behaviours. All sensor data is provided via a spike-based silicon retina camera (eDVS), and all control is implemented via spiking neurons simulated on neuromorphic hardware (SpiNNaker). Given this control system, the robot is capable of simple obstacle avoidance and random exploration. To train the robot to perform more complex tasks, we observe the robot and find instances where the robot accidentally performs the desired action. Data recorded from the robot during these times is then used to update the neural control system, increasing the likelihood of the robot performing that task in the future, given a similar sensor state. As an example application of this general-purpose method of training, we demonstrate the robot learning to respond to novel sensory stimuli (a mirror) by turning right if it is present at an intersection, and otherwise turning left. In general, this system can learn arbitrary relations between sensory input and motor behaviour.**

Keywords. adaptive systems, mobile robotics, neurocontrollers, neuromorphics, robot control

I. Introduction

A long-standing dream for robotics is to provide the same sort of intelligent, adaptive, and flexible behaviour that is seen in living biological systems. Furthermore, by creating systems that emulate biological adaptivity, we can investigate intelligence in a very broad sense, including capabilities that are not yet seen in current machine learning methods [1]. However, it is still an open research question as to how to use biological inspiration to construct improved methods of robotic control. There are few examples of neural networks being used for control [2] in a robotic domain. Typically, the hardware used is standard general-purpose computing, and the algorithms are machine-learning based. This means that, while they may take some high-level inspiration from biology, the algorithms themselves do not directly map to biological details.

Over the last few years, a bridge has been forming between neuroscience and robotics [3]. Ongoing developments in neuromorphic hardware design now provide novel computing resources that are more suitable to directly implementing the types of computations found in biological systems, thus facilitating the use of observations and data provided by the neuroscience community. Practically speaking, we can now implement large numbers of neurons on a very small power budget, making neural control a promising energy-efficient direction for mobile robot applications. Ideally, this could lead to flexible adaptive control of systems while staying within a limited power budget.

Our study looks to demonstrate such a flexible control system using neuromorphic hardware and neural-based adaptive control. The approach combines the SpiNNaker computing platform [4], [5], the Neural Engineering Framework (NEF) [6], and a series of robots developed at the Technische Universität München.

For energy efficient implementation given the hardware, the neuron model employed here is the standard *leaky integrate and fire* (LIF) model, where the weighted (ω) sum of input current I causes the voltage V to build up until some threshold is reached, at which time the neuron emits a discrete spike:

$$\tau_m \frac{dV}{dt} = -V + \sum \omega I \tag{1}$$

When this spike occurs, this causes post-synaptic current to flow into the neurons it is connected to, with each connection having its own weight ω and with the post-synaptic current $h(t)$ exponentially decaying over time, as shown in Fig. 1:

$$h(t) = e^{-t/\tau_s} \tag{2}$$

The goal of this work is to explore algorithms that can be usefully implemented given hardware that efficiently implements neural networks of this form. In particular, we note that living creatures have both genetic, low-level hard-wired reflexes and they are also capable of developing novel associations between stimuli and responses that are entirely context-dependent. Behavioural studies indicate that they can learn to

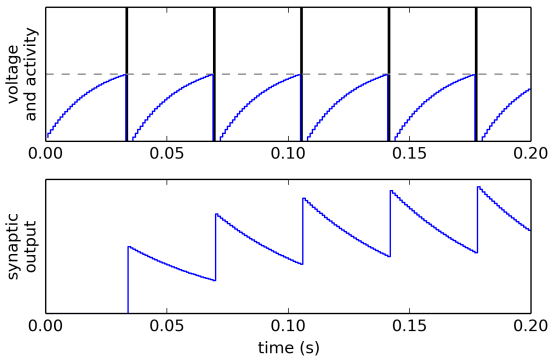

Fig. 1. Voltage V, spiking activity and output of a single LIF neuron, given a constant input I. SpiNNaker uses a simulation time step of $dt = 0.001$.

perform certain actions at certain times, through experience, overriding and building upon these low-level reflexes [7].

However, for applied robotics applications, we do not want an approach where learning is entirely autonomous and undirected (as in, for example, the neural learning seen in Distributed Adaptive Control [8]). Instead, our approach is to inform neural learning by providing explicit indications as to the instances where correct behaviour was achieved. The approach described here is somewhat akin to reinforcement learning, but relies only on positive examples, and can be explicitly shaped as desired. This guides the learning and provides explicit control over the eventual behaviour.

II. Infrastructure

A. eDVS

The sensor system used here is the eDVS embedded dynamic vision sensor [9], a silicon retina developed by iniLabs in collaboration with the Institute of Neuroinformatics at the University of Zurich and the ETH Zurich. This neuromorphic sensor is a 128x128-pixel camera. Instead of reporting frame-based data, it emits individual events when the relative brightness for any individual pixel increases or decreases. The eDVS provides high temporal resolution (∼1us), low latency (∼15us) and high dynamic range (∼120dB). The eDVS used here is an embedded version with an onboard microcontroller (NXP LPC4337), inertial measurement unit, multiple PWM control signals, and general-purpose I/O lines. The silicon retina is well-suited for integration with other neuromorphic hardware since it produces output in the form of spikes, which is the natural communication framework for spike-based neural processors. Furthermore, certain image processing algorithms can be implemented efficiently. For example, previous work [10] has implemented high-speed tracking of flashing LEDs, and we use that algorithm here as sensory pre-processing.

B. PushBot

At the Technische Universität München, the Neuroscientific System Theory group has developed a small tread-based robot built around the eDVS sensor, as shown in Fig. 2. The robot provides two motors, a frequency-controllable laser pointer, two LEDs, and housing for power (4 AA batteries). The setup comes with a proprietary WLAN module, enabling streaming of sensor data and motor command to and from the robot over standard WiFi to a SpiNNaker hardware interface board [11].

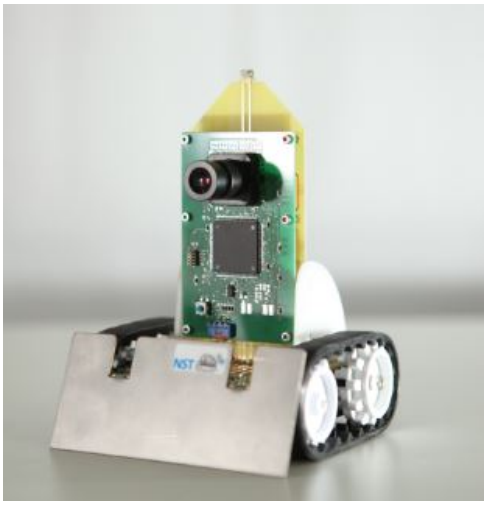

Fig. 2. The PushBot and the eDVS silicon retina.

C. SpiNNaker

The SpiNNaker multicore processor is developed by the University of Manchester and consists of 18 200MHz ARM968 cores on a single die [4], [5]. The processors and inter-chip communication infrastructure are optimized for massively parallel operations, with a target of a one-million-core machine. Its low power consumption means that a 48-chip SpiNNaker board with a total of 864 processors uses under 40W of power. This system can be programmed directly in C or using the standard neural modeling API PyNN. However, for this work we made use of the Nengo open-source neural compiler [12] (described below), and the custom Nengo backend that compiles high-level neural models into optimized SpiNNaker code [13].

D. Nengo and the Neural Engineering Framework

The Neural Engineering Framework (NEF) is a general-purpose neural compiler that allows the user to define a high-level algorithm that is then compiled down to a neural approximation of that algorithm [6]. This approach was originally meant for constructing complex biologically realistic neural models. Demonstrations of the use of Nengo for perception, cognition, and action encoding can be seen in Spaun, the first large-scale (2.5 million neuron) functional brain model capable of performing multiple tasks [14].

The basic concept of modelling using the NEF is the same as most neural network approaches: groups of neurons form distributed representations of their input, and the weighted sum of the output from neurons is used to approximate complex functions. For example, in Fig. 3, the six square sensor boxes are the six values returned via the sensory preprocessing on the PushBot robot, indicating the position (x, y) and certainty

(c) information of two flashing lights (the laser pointer dot and the LED on the top of the robot). This information is encoded into the sensory neurons using random connections, which ensures a diverse and rich "hidden layer" representation in the sensory neurons, allowing a wide range of output functions to be represented[1].

Therefore different patterns of activity within that group of neurons correspond to different vector values being represented. This vector is (normally) of much smaller dimensionality than the number of neurons in the group, so the neural firing forms a redundant code for that vector. Each neuron in the group responds differently to the same input, due to the random input connectivity. Therefore, the redundant code can be thought of as a random projection from a low-dimensional space (the vector being represented) to a high-dimensional space (the neural activity).

Connections out of a group of neurons implement functions on those represented vectors. That is, given some arbitrary function of the input vector, we can find output connections from the neurons that will approximate that desired function. Importantly, due to the redundant code, these functions do not have to be linear functions; rather, any function can be specified, and the system will find the optimal connections between neurons to approximate that function. The Neural Engineering Framework treats this as a least-squares minimization problem and finds the feed-forward linear synaptic connection weights between the individual neurons that will most closely approximate the desired nonlinear function. Furthermore, the NEF also indicates how recurrent connections can be found that will approximate any desired differential equation. For both feed-forward and recurrent connections, the accuracy of the approximation will be dependent on the number of neurons and the function being approximated.

As an example, consider a group of neurons storing three values: the x, y position of an object and a certainty measure c that the object is at that location. (This example comes from the output of the tracking algorithm for flashing LEDs [10]). In the NEF, we might use 100 neurons to form a distributed representation of these three values (using more neurons would improve the accuracy of the representation). Each neuron would be given a random combination of inputs from the three dimensions, resulting in a different pattern of firing for different vectors (x, y, c).

Given this activity, we can then define connections to other groups of neurons that compute functions of these values. For example, if we want a group of neurons R to store the distance from the center of the visual field to the location of the LED, we could optimise the connections so as to approximate:

$$\text{R} \leftarrow \sqrt{x_{LED}^2 + y_{LED}^2}$$

This would find synaptic connection weights between the group of neurons representing the LED data and the group of neurons representing the radius R such that R is driven to fire with whatever pattern represents the correct radius, given the current activity of the LED population.

All of the neural systems described below are defined in this manner, using the software package Nengo [12] developed specifically for such neural network construction, along with its interface for efficiently simulating such networks on SpiNNaker [13].

III. Method

The methodology used here is to start by building a neural system that implements a base set of automatic "reflex-like" behaviors for the robot. This is meant to correspond to the hardwired automatic responses found in living creatures. In addition to this, we will then add a learned system where the robot must map its sensory state to the particular desired actions for this condition. This corresponds to learned associative responses in animals.

A. Initial Reflexive Control

The first stage of this study requires a definition of the base set of simple behaviors and their triggering conditions, for a small mobile robot. These behaviors should be as simple as possible, since they must be hand-designed, but should provide a fairly rich repertoire of resulting actions. These can be thought of as the basic, instinctive reflexes seen in living creatures. In general, the first step in using this methodology is to define a collection of these reflexes.

The first reflexive behavior is simply to go forward if there is no obstruction. To define this, we specify a function that uses sensor data to compute the current strength S of an action (i.e. 0 to 1). In this case, the visual position of the dot caused by the laser pointer mounted on the robot can be used as a simple range detector: if it is lower than some threshold, the robot is near a wall and should not go forward. If it is higher than that threshold, it is safe to go forward:

$$\text{S[0]} \leftarrow 1 \textbf{ if } y_{LASER} > -0.6 \textbf{ else } 0$$

To complete this basic behavior, we define the motor output M as a function of S. In this case, we want to drive both wheel motors forward when the action has a high strength.

$$\text{M} \leftarrow [1,\ 1] \cdot \text{S[0]}$$

The next basic reflexive action is to back up when we are too close to an obstacle. This is again detected by using the visual location of the dot shown by the laser pointer. If this is very low in the visual field, or if it is not visible at all (i.e. the dot is below the field of view of the camera) then the robot should back up. This gives the following rule:

$$\text{S[1]} \leftarrow 1 \textbf{ if } y_{LASER} < -0.8 \text{ or } c_{LASER} < 0 \textbf{ else } 0$$
$$\text{M} \leftarrow [-1,\ -1] \cdot \text{S[1]}$$

The final basic actions are: turn left or right when close to an obstacle.

$$\text{S[2]} \leftarrow 1 \textbf{ if } y_{LASER} < -0.8 \textbf{ else } 0$$
$$\text{M} \leftarrow [-1,\ 1] \cdot \text{S[2]}$$

[1]One can use a learning rule such as backpropagation of error to optimize these weights, but we generally find this to be unnecessary

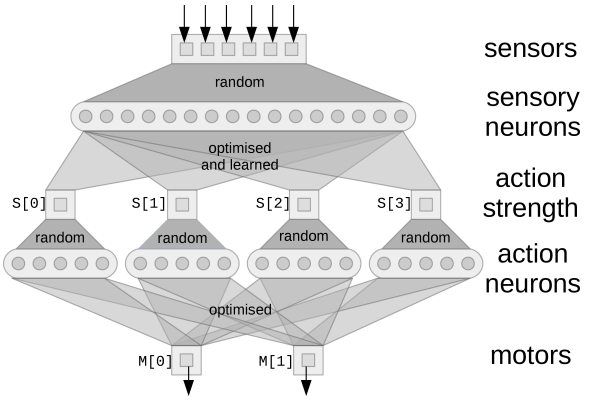

Fig. 3. A network implementing basic reactive control. Square boxes are the values being represented by the neurons (circles). Random connectivity ensures the neurons form a distributed representation of the vector values that are their input. The optimised output connections are solved for using least-squares minimization to approximate the functions listed in the text. Learned connections are added afterwards, as discussed in section III-B.

$$\text{S[3]} \leftarrow 1 \textbf{ if } y_{LASER} < -0.8 \textbf{ else } 0$$
$$\text{M} \leftarrow [1, -1] \cdot \text{S[3]}$$

However, these last two actions should not both be performed at the same time, since this would cause the robot to stay in place, rather than turning one direction or the other. To specify this, we can define connections that relate the strengths of different actions to each other.

$$\text{S[2]} \leftarrow -\text{S[3]}$$
$$\text{S[3]} \leftarrow -\text{S[2]}$$

This means that whenever $S[2]$ is positive, it will force $S[3]$ towards 0, and similarly $S[3]$ will drive $S[2]$ towards 0. This implements a classic neural competition, meaning that only one of the two actions will have a large value at a particular time.

Now that these reflex actions have been defined in these simple terms, we use Nengo to build neural networks that approximate each of these rules. Groups of neurons are defined to store the sensory state, the motor output, and the strengths of each action. The connections between the groups of neurons are set to approximate each of the above functions, resulting in the neural system shown in Fig. 3.

To understand the resulting basic reflexive behavior, two important aspects of NEF must be considered. First, due to the neural representation scheme, when there are multiple inputs to a neural population, the value represented will be the sum of the input values. In other words. the motor output will be the sum of all the connections: $S[0]$, $S[1]$, $S[2]$, and $S[3]$ to M. Second, the NEF approximations of the desired functions will be *smooth*. That is, instead of implementing the exact functions specified above, the actual behavior will be much more gradual.

For example, the input function to the $S[0]$ population is supposed to compute "1 if $y_{LASER} > -0.6$ else 0". Instead of implementing this exact function, the neurons will instead implement a smoothed approximation of this step function, as shown in Fig. 4.

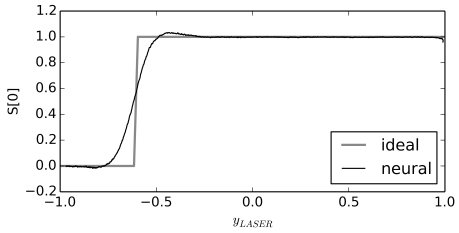

Fig. 4. Neural approximation with the NEF. When connections between neural groups are optimized to approximate a function, the result is a smooth version of that function. As the number of neurons is increased, this neural approximation will approach the ideal function.

The result is a small spiking neural network control system that performs very simple obstacle avoidance. Due to the smoothing inherent in the NEF neural approximation of the above functions, the robot gradually transitions between behaviors. This means that it automatically slows when approaching a wall, even though we have not explicitly built reflexes to do so.

When it reaches a wall, it turns in a random direction. This randomness is due to sensory noise, as there is no stochasticity in the neural model itself. Once it has chosen a particular direction to turn, i.e. once either $S[2]$ or $S[3]$ has a high value, it will continue to turn in that direction until there is no longer an obstacle in front of it. All actions transpire without human interference. The only hardcoded input are the initial thresholds that provide information as to whether an action is, in fact, correct given the current sensory state. These actions demonstrate obstacle avoidance and simple decision making, in much the same way as animal behavior studies [7].

The core idea is that this set of behaviors is a bare minimum, sufficient to allow the robot to move throughout its environment without human intervention. As with animal behaviour, this allows the robot to have a basic set of automatic decision-making rules that apply in novel situations where there is not yet any learned response. Using these reflexive rules, the robot makes "accidental" decisions (turning left in some situations and turning right in others, for example), which we will then use to further train the robot's behavior, as discussed in III-B.

B. Serendipitous Offline Learning

While the above approach is sufficient for implementing simple behaviors, it requires the developer to define explicit rules mapping sensory states to actions. In this section, we build upon these rules in a less explicit manner, allowing us to create more complex behavior without hard-coding any additional rules. In complex situations, it may not be feasible for action rules, such as the ones defined above, to be hard coded. Instead, we can also define *implicit* action rules. The basic idea is to allow the robot to explore its environment, but whenever it happens to accidentally do whatever action we want it to do, we record the sensory and motor values and *use those values to define a new rule*.

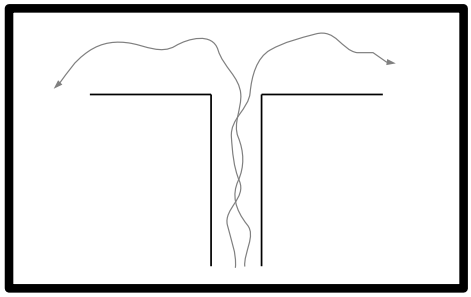

Fig. 5. The T-Maze environment, top-down view. The robot starts at the bottom of the T shape. A mirror is sometimes placed at the intersection (see IV-C).

For example, consider the simple case where we want the robot to always turn left at the intersection, rather than randomly turning either left or right. Recording the sensory data and the strength of each action S while the robot is performing its initial basic behavior, we find instances where the robot performs the desired action (turning left). We call these the *positive examples*. In this case, we consider the four individual runs shown in Fig. 6 where the robot's rotation happened to remain positive. The data from runs where it turned right are removed. The data from these positive examples can be thought of as long sequences of state-action pairs, indicating what action to perform in what state. Given this, we add new connections from sensors to the action strength populations. Instead of explicitly indicating what functions to approximate on these connections, we instead use the data gathered from the robot itself to be the target function that the neurons must approximate. This adjusts the neural control system, biasing the robot to performing similar actions in similar sensory states.

IV. Results

A. Initial behavior

To examine the model's behavior, we used a standard T-maze environment (Fig. 5). When placed at the bottom of the T-maze, the robot navigates forward, avoids the walls, reaches the choice point, and turns left or right. Typical trajectories are shown in Fig. 6, which indicates the motor outputs over time. Since the robot uses tank-style treads, the two motor output values are the left and right motor speeds, respectively. For clarity, here we plot the overall speed as the sum of M[0] (left motor) and M[1] (right motor) in the upper graph, while the rotation rate is the difference between the two, shown in the lower graph. Motor output values of typical individual runs are plotted along with an overall mean value (with 95% bootstrap confidence interval). All values are Gaussian filtered with $\sigma = 0.15s$ for visual clarity.

B. Learning Basic Responses

For the first and simplest learning task, we chose positive examples where the robot turned left at the intersection. The result of this training is shown in Fig. 7. Unlike Fig. 6, this

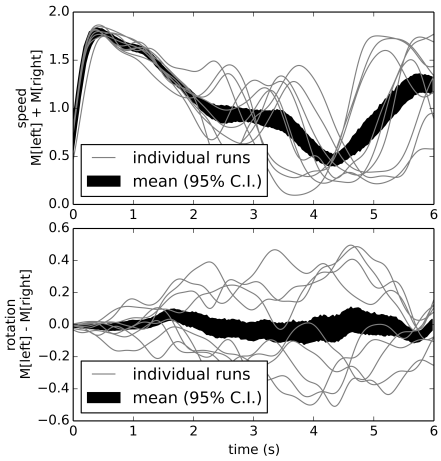

Fig. 6. Behavior of reactive control model over multiple runs. The speed (top graph) is high at the beginning, then slows as it turns either left or right (bottom graph). While on any individual run the robot tends to turn consistently either left or right, the overall average is zero turning (black area in bottom graph; area is 95% bootstrap confidence interval).

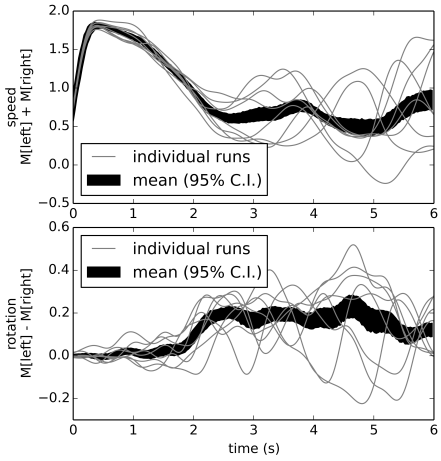

Fig. 7. Behavior after learning to turn left. By adding connections optimized to approximate situations where the robot behaved appropriately, we implicitly program the robot to map its sensory states to its actions as desired.

shows the robot consistently turning to the left about two seconds into the task (the average time for the robot to reach the end of the corridor). This indicates we can take a robot with one underlying reflexive control scheme and retrain it to change its behaviour without any explicit programming.

C. Learning Sensory Conditions

The initial example of learning to turn left is not particularly complex. However, we can use exactly the same process to produce more complex behavior. To demonstrate this, we now sometimes place a *mirror* at the intersection. Initially, the robot will ignore the mirror and just use its standard random reflexive navigation. We now choose as our positive examples situations where the robot turned right when there was a mirror, and situations where it turned left when there was no

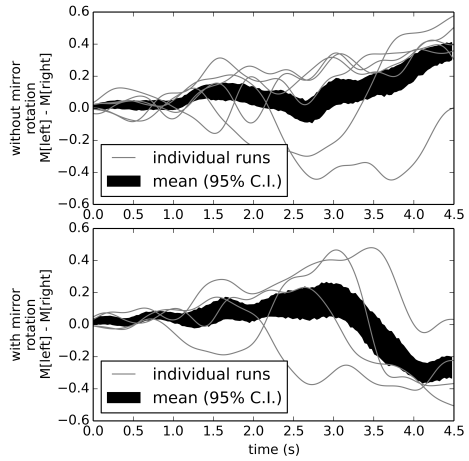

Fig. 8. Behavior after learning to turn right if there is a mirror, and otherwise turn left. The robot successfully identifies the correct situation and turns appropriately. Robot speed is not shown, but is similar to that depicted at the top of Fig. 6

mirror.

Adding a learned connection that attempts to approximate those positive examples means that the neural connections now need to change their outputs depending on whether the mirror is present or not. Fig. 8 shows that the robot can successfully learn to recognize and respond to mirrors, given only this simple learning rule on top of its basic reflexive rules.

V. Discussion

The algorithm described in this paper is a general-purpose approach to implementing mobile robot behaviours, making use of massively parallel low-power neuromorphic computing hardware. The basic algorithm is as follows:

1) define a set of basic actions (e.g. driving forward, turning left)
2) define functions for the strength of each action, given the current sensory state (e.g. drive forward if there is no obstacle ahead)
3) generate a neural network that approximates these functions
4) use the neural network to drive the robot and record the neural activity
5) identify a set of positive examples where the robot has accidentally performed whatever other task is desired (e.g. turning towards a target object)
6) retrain the neural connections between the sensory neurons and the action strengths so as to approximate the data gathered during the positive examples

Using this approach, we have shown that we can take a network that implements extremely simple manually specified reactive rules, and have it learn by example to perform more complex functions. This method makes use of the power-efficiency of neuromorphic hardware, where large numbers of neurons and connections can be used to efficiently approximate complex functions.

At first glance, it seems as if these new connections added during the training process (step 6) would just end up being exactly the same as the original reflexive control connections. However, the key difference here is that these new connections will also take into account *other sensory states*, not considered in the original hand-designed reflexive rules. In other words, these new connections will cause the robot to be more likely to perform the actions we desire whenever it is in a sensory state similar to those seen in the positive examples. Importantly, this will happen without any explicit indication of exactly what sensory states should trigger what actions. This allows the system to discover more complex rules than could be reasonably manually hand-designed.

In the case of the system learning to change its behavior based on the presence of a reflective mirror surface, the system was able to map the complex sensory stimuli to a particular motor action. In particular, it was able to make use of whatever sensory discrimination is available via the mirror (or lack of a mirror). It is important to note that this sensory stimulus did not initially impact the robot's behavior in any way. It was only through the process of learning, i.e. using hindsight examples of desired behavior and building a new set of connections that approximate required behavior that triggered changes in movement. This was done by building neural connections that replicate behaviour that the robot previously did accidentally.

A vital topic of future work is to characterize the variety of different learned tasks that are amenable to this method. In general, the NEF shows that the accuracy of the learned connections are dependent on the complexity of the function to be learned (in this case the mapping from sensor states to action choices) and the number of neurons [6]. As we explore different tasks, we will also need to explore different combinations of sensor systems and the pre-processing on this data.

This approach of using sensory experience to learn neural connections that attempt to cause the same actions to occur in response to similar sensory stimuli is novel, but we have performed previous work [15] with a somewhat similar learning method. In that case, we did not have an initial reflexive control system, and we did not find particular examples of desired behavior. Instead, we put the robot temporarily under manual control and used the behavior generated by the person performing the manual control to train the connections between sensory neurons and motor neurons. In other words, rather than allowing the robot to randomly explore its environment and choose actions based on a manually created reflexive control system, we had a user manually set the motor outputs using a remote control. As in this work presented here, we then trained neural connections that would approximate that manual control.

In that earlier work, learning required direct training examples, rather than simply labelling particular actions as positive example of desired behavior after they occur. Indeed, this means that the work presented here could be thought of as reinforcement learning, while the previous work would be purely supervised learning. Of course, a hybrid approach could

be pursued.

Finally, it should be noted that we are only training based on positive examples (i.e. situations where the robot behaved as desired). In future work, we plan to augment this approach with explicit punishment for situations where the robot performs a non-desired behavior. Interestingly, adding this capability is not straight-forward in this case. Indeed, there is significant biological evidence that positive (reward), and negative (punishment) systems are separate in the brain [TODO: reference]. That is, they are not simply the opposite of each other – rather they are significantly different processes that interact. This interaction is still to be explored. One step towards this is a model of fear conditioning [16], which would fit well with this framework.

ACKNOWLEDGEMENT

The Telluride Neuromorphic Cognition Workshop 2011.

REFERENCES

[1] D. McFarland and T. Bösser, *Intelligent behavior in animals and robots*. Mit Press, 1993.

[2] D. Janglová, "Neural networks in mobile robot motion," *Cutting Edge Robotics*, vol. 1, no. 1, p. 243, 2005.

[3] J. Krichmar and H. Wagatsuma, *Neuromorphic and Brain-Based Robots*. New York, NY, USA: Cambridge University Press, 2011.

[4] S. Furber and S. Temple, "Neural systems engineering," *Journal of the Royal Society interface*, vol. 4, no. 13, pp. 193–206, 2007.

[5] S. B. Furber, F. Galluppi, S. Temple, L. Plana *et al.*, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[6] C. Eliasmith and C. H. Anderson, *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press, 2004.

[7] Y. B. Kim, N. Huh, H. Lee, E. H. Baeg, D. Lee, and M. W. Jung, "Encoding of action history in the rat ventral striatum," *Journal of neurophysiology*, vol. 98, no. 6, pp. 3548–3556, 2007.

[8] P. F. Verschure, "Distributed adaptive control: a theory of the mind, brain, body nexus," *Biologically Inspired Cognitive Architectures*, vol. 1, pp. 55–72, 2012.

[9] J. Conradt, R. Berner, M. Cook, and T. Delbruck, "An embedded aer dynamic vision sensor for low-latency pole balancing," in *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 780–785.

[10] G. R. Müller and J. Conradt, "A miniature low-power sensor system for real time 2d visual tracking of led markers," in *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2429–2434.

[11] C. Denk, F. Llobet-Blandino, F. Galluppi, L. A. Plana, S. Furber, and J. Conradt, "Real-time interface board for closed-loop robotic tasks on the spinnaker neural computing system," in *Artificial Neural Networks and Machine Learning - ICANN 2013*, ser. Lecture Notes in Computer Science, V. Mladenov, P. Koprinkova-Hristova, G. Palm, A. Villa, B. Appollini, and N. Kasabov, Eds. Springer Berlin Heidelberg, 2013, vol. 8131, pp. 467–474.

[12] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. R. Voelker, and C. Eliasmith, "Nengo: a python tool for building large-scale functional brain models," *Frontiers in Neuroinformatics*, vol. 7, Jan. 2014. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3880998/

[13] A. Mundy, J. Knight, T. C. Stewart, and S. Furber, "An efficient SpiNNaker implementation of the Neural Engineering Framework," in *International Joint Conference on Neural Networks*, 2015.

[14] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, C. Tang, and D. Rasmussen, "A large-scale model of the functioning brain," *Science (New York, N.Y.)*, vol. 338, no. 6111, pp. 1202–1205, Nov. 2012.

[15] J. Conradt, F. Galluppi, and T. C. Stewart, "Trainable sensorimotor mapping in a neuromorphic robot," *Robotics and Autonomous Systems*, 2014.

[16] C. Kolbeck, T. Bekolay, and C. Eliasmith, "A biologically plausible spiking neuron model of fear conditioning," *Robotics and Autonomous Systems*, pp. 53–58, 2013.