Serendipitous Offline Learning in a Neuromorphic Robot

Terrence C. Stewart*, Andrew Mundy†, Ashley Kleinhans‡ and Jörg Conradt§
*School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, Georgia 30332–0250
Email: see http://www.michaelshell.org/contact.html
†Twentieth Century Fox, Springfield, USA
Email: homer@thesimpsons.com
‡Council of Scientific and Industrial Research, Pretoria, South Africa, akleinhans@csir.co.za
Telephone: (800) 555–1212, Fax: (888) 555–1212
§Tyrell Inc., 123 Replicant Street, Los Angeles, California 90210–4321

***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 initial 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 action we wish it to perform. 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. We demonstrate this learning approach is sufficient for the robot to learn to turn left or right depending on novel sensory stimuli.**

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

I. Introduction

[CHECK: represent related work and compare them with the proposed approach]

INTELLIGENT animal behaviour can be used for studies that mimic biological findings. The idea is to use recorded animalistic behaviour to guide robotic tools needed to facilitate understanding of real intelligence. By creating systems that emulate biological adaptivity, we can investigate the rational of intelligence in richer terms that what is provided by current machine learning methods [1]. Animal behaviour can be observed, and in invasive electrophysiology studies recorded, providing two complimentary sets of data to support further study. The open question is how best to make use of this data for robotic control. There are a few examples of neural networks being used for control [2] in a robotic domain. Typically the hardware is standard, and the algorithms are machine-learning based. Over the last few years, a bridge has been forming between neuroscience and robotics [3]. Ongoing developments in neuromorphic hardware design now provide the means to facilitate use of data provided by the neuroscience community. Practically speaking we can now use large numbers of neurones on a very small power budget making neural control a promising direction for mobile robot applications. Leading to flexible control, without significantly impacting battery life. Our study looks to leverage neuromorphic hardware and published animal behaviour studies. We examine an approach to this task combining: the SpiNNaker computing platform [4], [5]; the Neural Engineering Framework (NEF) [6] and a series of robots developed at the Technische Universität München. The neural model employed is that of the *leaky integrate and fire* (LIF), where: The neural activity is given as:

$$\alpha_i = G(\alpha_i \mathbf{e}_i \cdot \mathbf{x} + b_i) \tag{1}$$

The LIF model as [7] [CHECK: check notation and formula and reference please - is more explanation needed as per R1?]:

$$\alpha = \frac{1}{1 - \exp^{\frac{\tau_{ref} - \frac{1}{Hz}}{\tau_{rc}}}} \tag{2}$$

The goal of this work is to explore the usefulness of NEF using recet neuromorphic hardware, and platforms provided described in II. The particular algorithm of interest here is directly biologically inspired, and supported with behaviour studies. While living creatures have genetic, low-level hard-wired reflexes, they are also capable of developing novel associations between stimuli and responses that are entirely context-dependent. Behaviour studies indicate that they can learn to perform certain actions at certain times, through experience, overriding and building upon these low-level reflexes [8]. However, rather than allowing these associations to be learned entirely autonomously (as in approaches such as Distributed Adaptive Control [9]). Our approach is to inform learning by providing initial input that provides the robot with prior knowledge. Used to identify desirable situations, much like when an animal finds a food source and updates its situational awareness to return to that location. [CHECK: I think this is counter intuitive. We say that it does not do it autonomously, but then refer to it like its reinforcement learning, which is more autonomous, we need a clearer difference here between DAC and our system, we need to outline the reward

Fig. 1. The PushBot and the eDVS silicon retina.

signals - and make it clear what is updating and how this is novel]. This guides the learning and provides explicit control over the eventual behaviour.

September 3, 2015

II. Infrastructure

[CHECK R3: "proposed approach is not well organised and written, therefore cannot see valuable novelty in manuscript."]

A. eDVS

The sensor system used here is the eDVS embedded dynamic vision sensor [10]. 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 [11] 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 Figure 1. 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.

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, optimized for the 48-chip SpiNNaker board with a total of 864 processors using under 40W of power. This system can be programmed directly in C or using the standard neural modeling API PyNN. We made use of Nengo [12], an open-source neural compiler. A Nengo backend for SpiNNaker has been developed, which allows for testing of neural models on a standard PC, then recompiled such that they run natively on SpiNNaker.

D. NEF

The reason we chose Nengo rather than a more standard neural network framework is that Nengo [12] directly instantiates the Neural Engineering Framework (NEF). 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 meant for constructing complex biologically realistic neural models. Demonstration of the use of Nengo for perception, cognition, and action encoding can be seen in Spaun, the first large-scale (2.5 million neurons) functional brain model capable of performing multiple tasks [13]. When developing neural models using the NEF, every group of neurons represents a vector. 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 (neural heterogeneity). 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 between neural groups implement functions on those represented vectors. Importantly, due to the redundant code, these functions do not have to be linear functions; rather, any function can be specified. The Neural Engineering Framework treats this as a least-squares minimization problem and finds the linear synaptic connections between the individual neurons that will most closely approximate the desired nonlinear function. Furthermore (not used here), 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 store the resulting position of a flashing LED. A tracking algorithm takes the output of the eDVS camera and determines the x,y location (if any) of a flashing light at a particular frequency. The output is a three-dimensional vector (x, y, c). Where x and y are the pixel location of the flashing LED and c is a measure of certainty which should be 0 if no flashing at the desired frequency. In the NEF, we might use 100 neurons to form a distributed representation of these three values (more neurons would represent it more accurately). We can then define connections 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 LED, we could compute:

$$\mathrm{R} \leftarrow \textbf{sqrt}(\mathrm{led.x} \ast\ast 2 + \mathrm{led.y} \ast\ast 2)$$

This would cause the NEF to find synaptic connection weights between the group of neurons representing the LED data, and the group of neurons representing the radius $\ni$ the neural group `R` is driven to fire with whatever pattern represents the correct radius, given the current activity of the LED population.

III. METHOD

[TODO R2: provide more insight into methodology, to reflect scientific paper rather than technical report]

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 thought of as, the basic, instinctive reflexes seen in living creatures. For example, 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 laser pointer 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:

$$\text{S[0]} \leftarrow \textbf{if } \text{laser.y} > -0.6 \textbf{ else } 0$$

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. Similar to an animal moving forward only when there is sufficient space in front of it.

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

The next basic reflexive action is to back up when we are too close to an obstacle. This can be detected by the laser pointer threshold being too low. [CHECK: this says that if the laser is not visible then turn - is this correct??] If the laser pointer is so low that it is not visible, this action should also be triggered. Giving the following rule:

$$\text{S[1]} \leftarrow 1 \textbf{ if } \text{laser.y} < -0.8 \text{ or laser.c} < 0 \textbf{ else } 0$$
$$\text{M} \leftarrow [-1,\ -1] * \text{S[1]}$$

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

$$\text{S[2]}\ \ 1 \textbf{ if } \text{laser.y} < -0.8 \textbf{ else } 0$$
$$\text{M}\ \ [-1,\ 1] * \text{S[2]}$$

$$\text{S[3]}\ \ 1 \textbf{ if } \text{laser.y} < -0.8 \textbf{ else } 0$$
$$\text{M}\ \ [1,\ -1] * \text{S[3]}$$

These last two actions should not both be performed at the same time. To specify this, we can define actions that relate the strengths of different actions to each other.

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

Meaning whenever $S[2]$ is positive, it will force $S[3]$ towards 0, $S[3]$ similarly $S[2]$ towards 0. The result will be that only one of the two actions will occur at a time. Each of these basic actions can now be implemented using Nengo and NEF. Ensembles 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 set to approximate each of the above functions. Resulting in the neural system shown in Figure 2. To understand the resulting basic 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. I.e. 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 4 if $laser.y > -0.6$ else 0. The difference between this and the actual approximated function can be seen in Figure 3. 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. It will slow when approaching a wall, then turn in a random direction. This randomness will be due entirely 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. Rather than alternating back and forth between turning left and turning right. All actions transpire without human interference, or hard coding. The only hardcoded input are the initial thresholds that provide information as to whether an action is, in fact, correct. These actions demonstrate obstacle avoidance, and simple decision making, in much the same way as animal behavior studies [8]. To reiterate, it's important to note that no human interaction happens post initiation. [CHECK R2: Make clear that human observation is not required, and relate to animal behavior that, when first encountering a situation, makes "accidental" decisions, and then updates model of surroundings depending on outcome.] [TODO: Make clear that this means the system is learning, and the methodology is intrinsically automated - is there a way to describe this in math as suggested by reviewer 2??]

Fig. 2. [TODO R1: Terry fig's 5,6 and 7 should all show 10 trials (ie same amount of trails] Basic reactive control. Each box is implemented as a group of leaky integrate and fire [CHECK R1: explanation somewhere??] neurons, where N indicates the number of neurons, whose tuning curves span the dimensionality D as indicated. Each solid arrow indicates an alltoall connection with connection weights computed as per the NEF to approximate the functions described above. Dotted arrows indicate inputs and outputs connecting the robot to the neuromorphic hardware.

Fig. 3. 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.

Fig. 4. The T-Maze environment. The robot starts at the bottom.

Fig. 5. 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.

B. Initial behavior

To examine the model's behavior, we used a standard TMaze environment (Figure 4). 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 Figure 5, 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[left] and M[right], while the rotation rate is the difference between the two. 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.

C. 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 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. That is, we can use examples of the robot's behavior as the rules themselves. In particular, consider the simple case where we want the robot to turn left at the intersection, rather than randomly turning either left or right. Recording the sensory data and the strength of each action (S[0], S[1], etc.), 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 Figure 5 where the robot's rotation tended to remain positive and ignore the five runs where it was negative. The data from these runs can be thought of as long sequences of state-action pairs, indicating what action to perform in what state. Given this, we would be able to add new connections from sensors to the action strength populations, suggested for further study. Instead of explicitly indicating what functions to approximate on these connections, we instead use the NEF to find the connection weights that produce outputs that most closely match the original behavior. At first glance, it seems as if these new connections 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.

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).

Fig. 7. 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 Figure 6

IV. Results

The result of this training is shown in Figure 5. Unlike Figure 6 this 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).

[TODO R1: Look at adding a little more info about the data already recorded, some metrics, like information available vs number of trials kind of thing.]

A. Learning Sensory Conditions

The initial example of learning to turn left is not particularly complex. However, exactly the same method can be used for more complex cases. For example, we can place complex visual stimuli at the intersection, and chose the positive examples, i.e. situations where the robot turned left for one stimulus or right for another. The particular stimulus we used was the reflection of a mirror. Figure 7 shows that the robot can successfully learn to turn right when it sees a mirror, but turn left when there is no reflecting mirror.

V. Discussion

The algorithm described in this paper is meant to be a general-purpose approach, making use of massively parallel low-power neuromorphic computing hardware on a mobile robot. We have shown that we can start be defining extremely simple reactive rules, using NEF to implement them on the neuromorphic hardware. However, these sorts of simple reactive rules could, of course, be implemented using more traditional computing hardware. The point of this paper is to demonstrate automatic training of a simple set of neural connections that interacts with basic reactive systems to produce much more complex control. In the case of the system learning to change its behavior based on the presence of a reflective surface, the system was able to map the complex sensory stimuli to a particular motor action. While we have not yet completed a full analysis of the learned behavior, it is strongly aided by the presence of the flashing LED on top of the robot, which would then be visible. 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. Meaning that the scope of possible sensorimotor mappings is limited only by the range of the sensory inputs and the size of the neural population representing the sensors. If there are sufficiently many neurons

representing the sensory space, then NEF guarantees that connection weights can be found with a desired level of accuracy. [CHECK: I think this sentence should be removed - we are not looking at *any fxn* and should not make this overall claim: that will approximate any function with any desired, unless true and verified] . However, for extremely complex functions, the number of neurons required may be excessive. Thus, further investigation is required to characterize a set of different useful functions to determine how many neurons are needed. Note that this method of utilizing neural hardware, and software, does not have the problems seen by traditional neural network learning rules. Such as the back-propagation of an error [TODO: explain this statement with reference]. In particular, the only learning involves optimizing a single layer of connection weights. This task can be performed by any gradient-descent method, or by simple algebraic least-squares minimization. [CHECK: confirm correctness of this sentence] Unlike these methods, NEF is not subject to the problems of being stuck in local minima. Meaning that the same algorithm described here, using only 500 neurons to represent sensory state will scale up to situations where we use multiple SpiNNaker chips or even a full SpiNNaker board simulating approximately one million neurons. We will be characterizing these capabilities in future work. In previous work [14], we have used 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. That work is more restrictive than the model presented here since there is no initial scaffolding (the reflexive actions) for the learned actions to build on. Furthermore, that method requires direct training examples, rather than an after-the-fact manual labeling of particular instances as desired behavior. 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). We could augment this work with explicit punishment for situations where the robot performs a non-desired behavior. Adding this is not straight forward. We would need to modify the network to do anything, except a previous action, which is difficult to describe in a function. Interestingly, 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. NEF has already been used to model fear conditioning [15], which would fit well with this model. [CHECK: should we remove this sentence - is it not reactive modeling? rather than fear, which is in the sympathetic nervous system - is this the same system here?]

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] B. Webb, "What does robotics offer animal behaviour?" *Animal behaviour*, vol. 60, no. 5, pp. 545–558, 2000.

[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] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[8] 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.

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

[10] 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.

[11] 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.

[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] 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.

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

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