Serendipitous Offline Learning in a Neuromorphic Robot

Terrence C. Stewart*, Andrew Mundy,†, James Kirk‡, 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
‡Starfleet Academy, San Francisco, California 96678-2391
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 behaviors. 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

ONGOING developments in neuromorphic hardware design mean that it is possible to simulate large numbers of neurons on a very small power budget. This makes neural control a promising direction for mobile robotics applications, hopefully leading to flexible control without significantly impacting battery life. The open question is how best to make use of neural networks for robotic control. Our research program examines one possible approach to this task which combines the SpiNNaker computing platform [1], [2], the Neural Engineering Framework method for constructing neural models [3], and a series of robots developed at the Technische Universität München. The goal is to explore the new types of control algorithms that are made possible by neuromorphic hardware. The particular algorithm of interest here is directly biologically inspired, as is the task to be performed. While living creatures are genetically endowed with low-level hard-wired reflexes, they are also capable of developing novel associations between stimuli and responses which are entirely context-dependent. In other words, they can learn through experience to perform certain actions at certain times, overriding and building upon these low-level reflexes. However, rather than allowing these associations to be learned entirely autonomously (as in approaches such as Distributed Adaptive Control [4]), our approach is to have the programmer shape the learning by identifying particular situations where there robot performed a desired action correctly. This guides the learning and provides explicit control over the eventual behaviour.

August 18, 2015

II. Infrastructure

A. eDVS

The sensor system used here is the eDVS embedded dynamic vision sensor [5]. This is 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 128 x 128 pixel camera, but instead of reporting frame-based data, it emits individual events whenever the relative brightness for any individual pixel increases or decreases by a certain amount. This provides high temporal resolution (1us) and low latency (15us), as well as a 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 [6] has implemented high-speed tracking of flashing LEDs, and we we use that algorithm here as sensory pre-processing.

B. PushBot

At the Technische Universität München, the Neuroscientific System Theory group has developed a sma ll tread-based robot built around the eDVS sensor, as shown in Figure 1. This provides two motors, a frequency-controllable laser pointer, two LEDs, and housing for power (4 AA batteries). They have also developed a WLAN module, enabling streaming of sensor

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

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 [1], [2]. The processors and inter-chip communication infrastructure is optimized for machine, which is a 48-chip SpiNNaker board with a total of 864 processors using under 40W of power. While this system can be programmed directly in C or using the standard neural modelling API PyNN, we made use of Nengo [7], an open-source neural compiler, for which we have developed a custom backend for SpiNNaker. This allows neural models to be initially tested at small scales on a standard PC, and then recompile the models such that they run natively on SpiNNaker.

D. Neural Engineering Framework

The reason we chose Nengo rather than a more standard neural network framework is that Nengo [7] directly instantiates the Neural Engineering Framework (NEF), a general-purpose neural compiler that allows the user to define a high-level algorithm which is then compiled down to a neural approximation of that algorithm [3]. This approach is meant for constructing complex biologically realistic neural models that cover perception, cognition, and action, and it was recently used to build Spaun, the first large-scale (2.5 million neurons) functional brain model capable of performing multiple tasks [8]. 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. Since each neuron in the group responds differently to the same input (neural heterogeneity), this 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 (although it is not used here), the NEF also indicates how recurrent connections can be found that will approximate any desired differential equation. Of course, 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 of use, consider a group of neurons storing the result of the flashing LED tracking algorithm that takes the output of the eDVS camera and determines the x,y location (if any) of a flashing light at a particular frequency. This 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 is found. In the NEF, we might use 100 neurons to form a distributed representation of these 3 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, such that the neural group `R` would be driven to fire with whatever pattern represents the correct radius given the current activity of the LED population.

III. Method

A. initial reflexive control

The first stage is to define a base set of simple behaviours for the robot, along with triggering conditions for these behaviours. These behaviours should be as simple as possible, since they must be hand-designed, but should also provide a fairly rich repertoire of resulting actions. These can be thought of as the basic, instinctive, genetically-endowed reflexes seen in living creatures. For example, the first reflexive behaviour is simply to go forward if there is nothing in front of the robot. To define this, we specify a function that uses the sensor data to compute the current strength S of this action (from 0 to 1). In this case, the visual position of the laser pointer dot can be used as a simple range detector; if it is lower than some threshold the robot is near a wall and should to go forward. If it is higher than that threshold, it is safe to go forward.

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

To complete this basic behaviour, we define the motor output M as a function of the strength S of the action. 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.

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

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 position being too low. Indeed, if the laser pointer is so low that it is not visible, this action should also be triggered. This gives the following rule

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

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

Fig. 2. Basic reactive control. Each box is implemented as a group of LIF 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.

S[2] 1 **if** laser.y < -0.8 **else** 0
M $[-1, 1] * S[2]$

S[3] 1 **if** laser.y < -0.8 **else** 0
M $[1, -1] * S[3]$

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

$S[2] \leftarrow -S[3]$
$S[3] \leftarrow -S[2]$

This means that whenever S[2] is positive, it will drive S[3] towards 0, and S[3] will similarly drive 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 the Neural Engineering Framework. 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 set to approximate each of the above functions. This results in the neural system shown in Figure III-A. To understand the resulting basic behaviour, two important aspects of the Neural Engineering Framework 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. This means, for example, that the motor output will be the sum of all the connections from 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 behaviour will be much

more gradual. For example, the input function to the S[0] population is supposed to compute 1 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, and 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. Furthermore, 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

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

Fig. 5. Behaviour 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).

(rather than alternating back and forth between turning left and turning right).

B. Initial Behaviour

To examine the model's behaviour, 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 behaviours, 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 explicitly programmed. Instead, we can also define *implicit* action rules. That is, we can use examples of the robot's own behaviour 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. If we record the sensory data and the strength of each action (S[0], S[1], etc.) while the robot is performing its basic initial behaviour, we can find particular instances where the robot happened to perform the desired action (turning left). We call these the*positive examples*. In this case, we would 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 can 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 NEF to find the connection weights that produce outputs that most closely match the original behaviour. 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* that were not considered in the original hand-designed reflexive rules. In other words,

Fig. 6. 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.

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.

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. The result of this training is shown in Figure 6. Unlike Figure 5, this shows the robot consistently turning to the left about 2 seconds into the behaviour (the typical time at which the robot reaches the end of the corridor).

D. 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 as situations where the robot turned left for one stimulus or turned right for another stimulus. The particular stimulus we used was 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 mirror.

IV. Discussion

The algorithm described in this paper is meant to be a general-purpose approach for 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 the Neural Engineering Framework 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 that we can automatically train a simple set of neural connections that interacts with these 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 mirror in front of it, the system was able to successfully 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 helped by the presence of the flashing LED on top of the robot, which would then be visible in the mirror. However, 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 taking examples of desired behavior and building a new set of connections that approximate that behavior that this sensory stimuli came to trigger changes in movement. This means 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. That is, if there are sufficiently many neurons representing the sensory space, then the Neural Engineering Framework guarantees that connection weights can be found that will approximate any function with any desired level of accuracy. However, for extremely complex functions, the number of neurons required may be astronomical. We thus need to perform further work to characterize a set of different useful functions to determine how many neurons are needed. It should be noted, however, that this method of using neural hardware does not run into the problems seen by traditional neural network learning rules, such as backpropagation of error. 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. These are not subject to the problems of being stuck in local minima. This means that the same algorithm described here where there are only 500 neurons representing 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 [9], 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 upon. Furthermore, that method requires direct training examples, rather than an after-the-fact manual labeling of particular instances as desired behavior. Metaphorically speaking, the work presented here could be thought of as closer to 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 in this work we are only training based on positive examples (i.e. situations where the robot behaved as desired). It may also be possible to supplement this work with explicit punishment for situations where the robot performs non-desired behaviour. However, exactly how to do this is not straightforward. In particular, rather than using the NEF to approximate whatever the robot had done during the positive example, we would need instead to modify the network to do anything except what it had done. This is not straightforward to express as a function. Interestingly, there is significant biological evidence that positive (reward) and negative (punishment) systems are separate in the brain. That is, they are not simple the opposite of each other rather they are significantly different processes that interact. This interaction will also be explored in our future work, and the NEF has already been used to model fear conditioning [10], which would fit well with this model.

Acknowledgement

The authors would like to thank...

REFERENCES

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

[2] 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.

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

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

[5] 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.

[6] 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.

[7] 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/

[8] 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.

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

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