Serendipitous Offline Learning in a Neuromorphic Robot

Terrence C. Stewart, Andrew Mundy, Ashley Kleinhans, and Jörg

***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.**

***Index Terms*—Enter key words or phrases in alphabetical order, separated by commas. For a list of suggested keywords, send a blank e-mail to keywords@ieee.org or visit** http://www.ieee.org/organizations/pubs/ani_prod/keywrd98.txt

I. Introduction

THIS is an introduction. It should have some text in it and talk about what we're trying to do and why and what we've done before. It should probably also mention DAC and indicate why this is different.

II. Infrastructure

Overview of hardware and software. Include a list of sensors and motors on the robot, and a list of just the ones used in this paper (and the notation we will use to refer to them below).

III. Method

A. Initial Reflexive Control

The first stage is to define a base set of simple behaviors for the robot, along with triggering conditions for these behaviors. These behaviors 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 behavior 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.

```
S[0] ← 1 if laser.y > -0.6 else 0
```

To complete this basic behavior, 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.

```
M ← [1, 1] * 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

```
S[1] ← 1 if laser.y < -0.8 or laser.c < 0 else 0
M ← [-1, -1] * S[1]
```

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

```
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] ← -S[3]
S[3] ← -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 1.

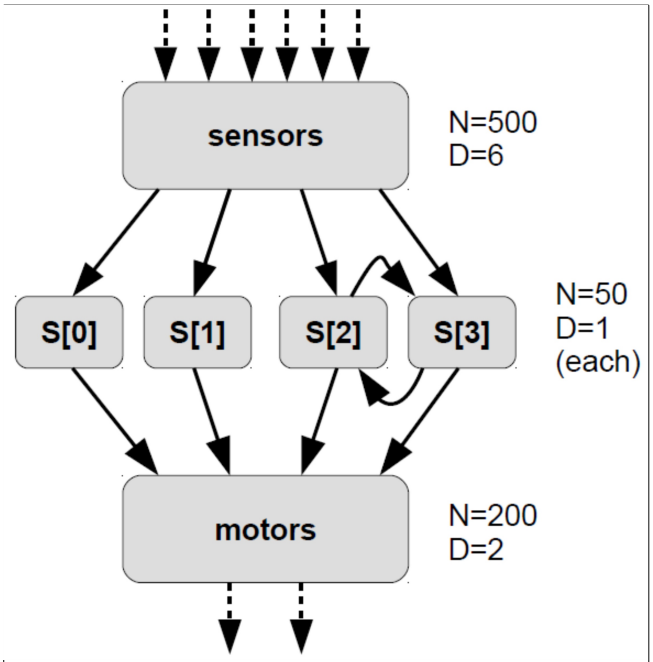

Fig 1. Basic reactive control. Each box is implemented as a group of LIF neurons, where N indicates the number of neurons, whose tuning curves span the dimensionality D as indicated. Each solid arrow indicates an all-to-all connection with connection weights computed as per the NEF to approximate the functions described above. Dotted arrows indicate inputs and outputs connecting the robot to the neuromorphic hardware.

To understand the resulting basic behavior, two important

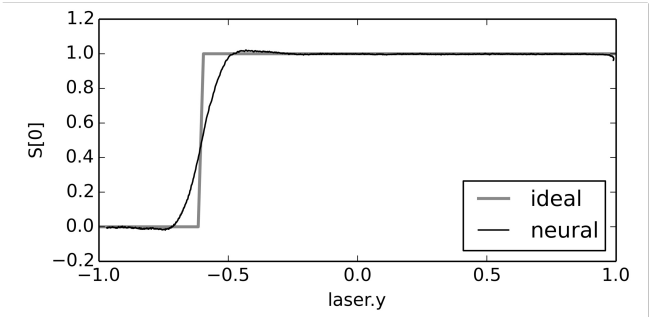

Fig 1. Neural approximation with the NEF. When connections between neural groups are optimized to approximate a function, the result is a smooth version of that function. As the number of neurons is increased, this neural approximation will approach the ideal function.

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 behavior 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 2.

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

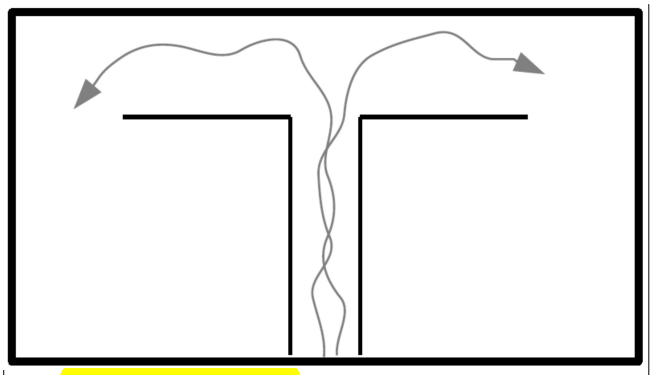

Fig 1. The T-Maze environment. The robot starts at the bottom.

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

B. Initial Behavior

To examine the model's behavior, we used a standard T-Maze environment (Figure 3). 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 3.

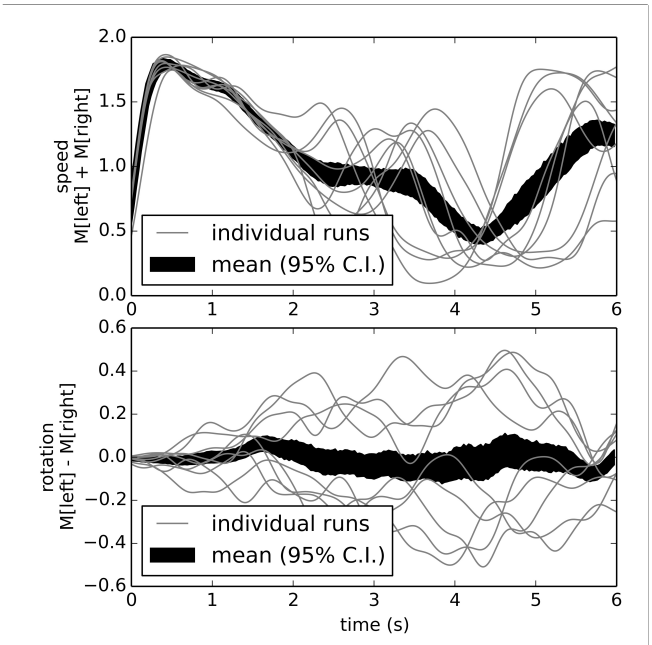

Fig 1. Behavior of reactive control model over multiple runs. The speed (top graph) is high at the beginning, then slows as it turns either left or right (bottom graph). While on any individual run the robot tends to turn consistently either left or right, the overall average is zero turning (black area in bottom graph).

Figure 4 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 ares Gaussian filtered with σ=0.15s.

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 explicitly programmed. Instead, we can also define *implicit* action rules. That is, we can use examples of the robot's own 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. 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 behavior, 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 4 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 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* that were 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.

The result of this training is shown in Figure 5. Unlike Figure 4, this shows the robot consistently turning to the left about 2 seconds into the behavior (the typical time at which the robot reaches the end of the corridor).

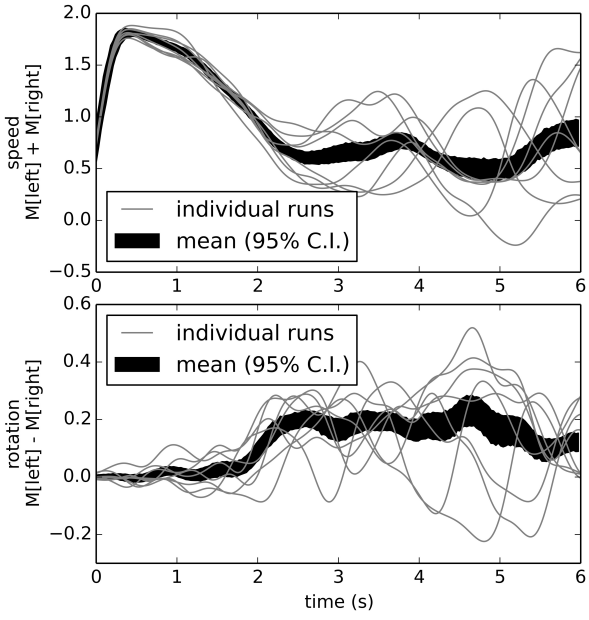

Fig 1. Behavior after learning to turn left. By adding connections optimized to approximate situations where the robot behaved appropriately, we implicitly program the robot to map its sensory states to its actions as desired.

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 6 shows that the robot can successfully learn to turn left when it sees a mirror, but turn right when there is no mirror.

[figure 6]

IV. Discussion

[the mirror example is helped by identifiable LED on the robot]
[but arbitrarily complex mapping should be doable as the number of neurons in the sensory population grows]
[only limited by what sensor values are fed in there]
[compare to learning-by-direct control example in previous paper]