Short-Term Plasticity in a Liquid State Machine Biomimetic Robot Arm Controller

R. de Azambuja*†, F. B. Klein*, S. V. Adams*, M. F. Stoelen*, A. Cangelosi*

*School of Computing, Electronics and Mathematics
University of Plymouth
Plymouth, Devon, United Kingdom

†CAPES Foundation
Ministry of Education of Brazil
Brasilia, DF 70040-020, Brazil

Email: {ricardo.deazambuja, frederico.klein, samantha.adams, martin.stoelen, a.cangelosi}@plymouth.ac.uk

Abstract

Biological neural networks are able to control limbs in different scenarios, with high precision and robustness. As neural networks in living beings communicate through spikes, modern neuromorphic systems try to mimic them making use of spike-based neuron models. Liquid State Machines (LSM), a special type of Reservoir Computing system made of spiking units, when it was first introduced, had plasticity on an external layer and also through Short-Term Plasticity (STP) within the reservoir itself. However, most neuromorphic hardware currently available does not implement both Short-Term Depression and Facilitation and some of them don't support STP at all. In this work, we test the impact of STP in an experimental way using a 2 degrees of freedom simulated robotic arm controlled by an LSM. Four trajectories are learned and their reproduction analysed with Dynamic Time Warping accumulated cost as the benchmark. The results from two different set-ups showed the use of STP in the reservoir was useful for one out of three tested trajectories, though not computationally cost-effective for this particular robotic task.

Index Terms

Liquid State Machine, Reservoir Computing, Short-Term Plasticity, Robotics.

I. Introduction

The human brain could be seen as a complex, non-linear and parallel cognitive machine. Also according to Taylor [1], an autonomous agent needs to have at least some basic components as attention, memory, motor control and emotions. All these emerge from the dense networks formed by the brain's neurons. Artificial Neural Networks (ANNs) are implementations (software or hardware based) inspired by the discoveries made throughout the last hundred years about the brain.

In the work of Maass [2], ANNs are classified into three different generations: first, second and third. The first generation is composed of binary gates widely known as perceptrons or McCulloch-Pitts neurons. They are considered universal for digital computations and every boolean function can be implemented using the proper network structure. More complex neuron models using continuous activation functions define the second generation.

The continuous behaviour of the implemented neuron gave rise to the discovery of the revolutionary back-propagation training method as gradient descent demands a continuously differentiable function (see [3] for a historical overview).

As real neurons send and receive information using pulses (spikes), the second generation of artificial neural networks uses the biological interpretation of continuous activation as a firing rate (number of spikes produced in a given time window). This type of code could become problematic when fast computations are considered, since firing rates are mean values and, in a very short time window, a meaningful rate cannot be defined [4].

The third generation of neural networks employs spiking neurons (e.g. integrate and fire neurons) as its basic unit. It is also known as artificial Spiking Neural Networks (SNNs). Maass [2] states the third generation is computationally more powerful than the first two generations (when the necessary number of neurons required for a certain task is considered) and on top of that, the spiking neuron has, in the worst scenario, the same computational power. Also, according to Adams et al. [5], there is evidence from neurobiology suggesting spikes are important for cognitive and behavioural tasks.

Naturally occurring SNNs make use of internal plasticity in the connections between neurons to learn or adapt. However, as an artificial implementation scales up and multiple recurrent connections are used to increase the power at processing time series, this distributed plasticity brings a high computational cost. A solution for this problem is the use of a spiking neuron based reservoir computing system where the long-term plasticity can be isolated to an external layer called *readout*. Such a system is known as a Liquid State Machine (LSM). Besides the possibility of having plasticity only on the *readout*, when introduced [6] LSMs were actually designed with the internal use of an implementation of Short-Term Plasticity (STP). STP is the dynamical change in the synaptic efficacy over time. It can also be compared to dynamical memory buffers [7]. According to Rotman et al. [8] STP increases the transmission of information in the frequency range of the naturally occurring spikes. According to Maass et al. [9], the LSM parameters applied to the neuron model, connection probability and STP were biologically plausible coming from real measurements of microcircuits in rat somatosensory cortex.

Going back to Taylor's work [1], there is a necessity of developing hardware to provide a physical realisation of the neuron processing in the brain. In the later years, neuromorphic hardware implementations are becoming an important tool to simulate complex SNNs. The best known systems are totally digital (SpiNNaker [10] and TrueNorth [11]) or a mixed analog and digital implementation (Neurogrid [12], Spikey [13] and ROLLS [14]). All those neuromorphic systems can implement plasticity to a certain level. Still, only ROLLS seems to implement simultaneously Short-Term Depression and Facilitation. Whilst SpiNNaker, TrueNorth and Neurogrid can carry out some long-term plasticity in the form of Spike-Timing Dependent Plasticity (STDP), currently, they are not equipped with STP implementations. Spikey is capable of implementing Short-Term Depression or Facilitation, but not both for the same synapse.

The next sections present results from a biomimetic robot arm controller based on a spiking neural reservoir computing system, the Liquid State Machine, as a benchmark task for testing the effectiveness of Short-Term Plasticity. A simulated two degrees of freedom robotic arm plays the role of a physical body providing the proprioceptive feedback and embodiment to our artificial spiking neural network.

II. Methods

In order to verify the effectiveness of STP inside an SNN based reservoir, a robotic task comprised of learning to reproduce four distinct trajectories was defined (Fig. 4) and a biomimetic robot arm controller was implemented using an LSM with extra proprioceptive feedback connections from the readout to the inputs. The experiments were executed controlling a 2 degrees of freedom simulated robot arm (Fig. 1). Two different set-ups were used where each experiment set had a total of 50 trials for each trajectory. The generated trajectories were analysed with the Dynamic Time Warping and the results statistically verified using the Welch's t-test.

The idea of an LSM based arm controller, like the one implemented here, was first introduced by Joshi and Maass [15] and it was used as the base for this implementation. However, the system presented by the authors [15] doesn't have source code available, making our system the only one open source and readily accessible to any researcher (See Sect. IV for links).

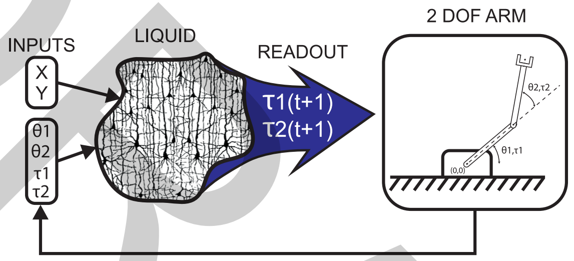

Figure 1. Illustrative representation of the arm controller. X,Y are the coordinates of the final position, $\tau1$ and $\tau2$ the commanded torques and $\theta1$ and $\theta2$ the current joint angles (proprioceptive feedback).

A. Liquid State Machine Simulation

The Liquid State Machine (LSM), first proposed by Maass et al. [6], is a neural computation framework where one or more input signals are injected in a non-linear dynamical system (a group of artificial neurons modelling a column in the cortex) and the disturbances generated are collected and interpreted in the output (readout). Also, the authors [6] have demonstrated the universal computational power of this model when some idealized conditions, separation and approximation properties, are met. A simple diagram representing an LSM can be seen at the center of Fig. 1. This framework contrasts with the Turing machine paradigm because the states generated are not sequential or even stable. However, the LSM has in its perturbed states all the information representing the current and past inputs (in theory the fading memory never totally vanishes, but in practical terms the past inputs last a finite time) which makes it an interesting tool to process time-series data as can be seen in the literature [16]–[18]. All the computations being accomplished by the system are easily accessed using a linear classifier on the readout neurons such as a least-squares linear regression or even a perceptron. Schürmann et al. [19] justify the use of such simple classifiers stating that the high-dimensional space created by the liquid increases the probability of having a linearly separable classification problem at the end.

An LSM usually is composed of Leaky Integrate-and-Fire (LIF) neurons [20] (see Section II-A1) connected in a recurrent pattern as suggested in the literature [6] forming what is known as Small-World Network (SWN). Basset and Bullmore [21] suggest this type of network presents an appealing way to model the brain connections based on empirical and theoretical motivations. Compared to models where an All-to-All connection methodology is used, SWN can make use of far less wiring yet able to generate high dynamical interactions through many indirect feedback loops when a properly chosen probability connection is used. According to Watts and Strogatz [22] SWN combines high clustering ability with a short path length.

In addition to the biologically plausible neuron model and network parameters this approach implements noise levels that account for what is found *in vivo* recordings [23]. It is known, as well, that stochastic processes are very important for brain functioning [24] and several noise sources are actuating inside the brain [25]. Moreover, in the work of Dockendorf et al. [26] was proved that some principles of LSM could be found within *in vitro* neural networks of cortical neurons making clear it's not only computationally useful, but could help to explain how biological neural systems work too.

The LSM, as originally defined [6], was considered to have the main limitation related to the extension of its fading memory and only computations requiring integration in a time range of about $300ms$ (this value actually changes according to the parameters used) were possible. In summary, the LSM was not able to be applied to tasks with long duration. However, it was proved [27] that if a feedback loop is used, the LSM becomes capable of emulating arbitrary Turing machines. The same authors also extended this idea stating that ”any dynamical system” can be simulated with appropriate feedback. In addition, they declared such systems can still perform in the presence of realistic noise levels, but they have the computational power reduced.

In a feedback system like the one presented here, according to Hauser et al. [28], the use of noise is crucial during the learning phase to make the system robust to small variations in the feedback values during the testing phase. Therefore, a discrete noise source was used directly in the inputs (depending on the experiment set, some inputs don't receive noise). A value drawn from a uniform distribution (Numpy method *randint*) going from $min = -1$ to $max = 1$ was used. That way, it is possible to have a better control over the input spikes during the training phase and the signal-to-noise ratio is automatically made proportional to the signal range. Also, because the feedback during the testing phase is made of spikes instead of analogue values, this discrete noise mimics the real testing situation.

The liquid was created using 600 LIF neurons (forming a 3D structure of size $20 \times 5 \times 6$, see Fig. 2) where 80% were excitatory and 20% inhibitory. Its internal connections were generated with a probability according to the distance between neurons ($D(a, b)$) and their types (excitatory - E or inhibitory - I) generated by Equation 1 [6], where the λ value was kept constant and equal to 1.2. The constant C had its value changed according to the type of neuron: 0.3 (EE), 0.2 (EI), 0.4 (IE) and 0.1 (II).

$$P_{conn} = Ce^{-\frac{D(a,b)^2}{\lambda^2}} \tag{1}$$

In the work of Joshi and Maass [15], the system had an input layer made of LIF neurons, whilst here this layer is abstracted and the values are injected directly into the liquid, following a Gaussian distribution of weights, that works like a receptive field. The input layer was implemented as 300 virtual neurons subdivided into 6 groups. As a sub-layer after the main liquid, there are 600 membrane low-pass filters (one individual filter for each neuron inside the liquid) processing the spikes before they are sent to the readout. The final structure of the liquid used through all the experiments can be seen in the Fig. 2. Since the liquid has a small-world connection pattern, it is important to highlight the formation of short and long range connections and the way they are distributed.

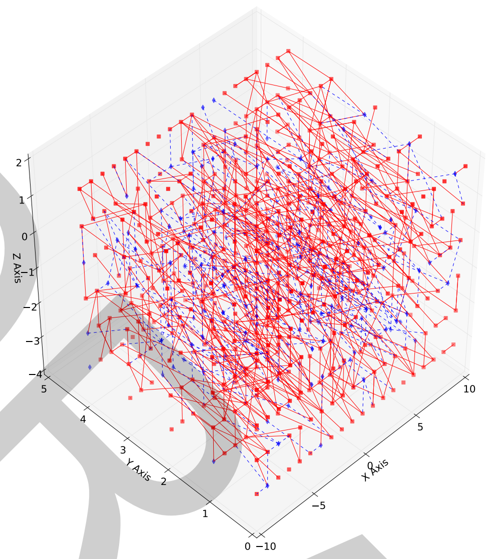

Figure 2. Visualization of the liquid's shape (showing only the connections between two different neurons) used for all experiments. Red squares indicate the excitatory neurons whilst blue diamonds the inhibitory ones. Continuous red lines are excitatory connections and dashed blue ones inhibitory. Self-connections are not displayed. The parameters used to generate the liquid's structure can be seen in Table II.

The virtual input layer is composed of 300 neurons, but, as they resemble a simplified version of a population code with 6 variables, only 50 unique neurons are available to represent each variable. A lookup table was created for each variable. Hence to simplify the conversion, an offset was added to the final tables to adjust them to the respective neuron group. Each group was associated to one of the input variables as following: 1) X-Coordinate of the end position: linear distributed values between -1 and 1 and no offset. 2) Y-Coordinate of the end position: linear distributed values between -1 and 1 with an offset of 50. 3) Joint 1 - proprioceptive angle: linear distributed values between $-\frac{\pi}{6}$ and π with an offset of 100. 4) Joint 2 - proprioceptive angle: linear distributed values between 0 and π with an offset of 150. 5) Joint 1 - proprioceptive torque: linear distributed values between the maximum ($9.93Nm$) and minimum ($-11.93Nm$) ones that occurred during the training phase with an offset of 200. 6) Joint 2 - proprioceptive torque: linear distributed values between the maximum ($3.35Nm$) and minimum ($-2.30Nm$) ones that occurred during the training phase with an offset of 250.

The conversion of the analogue values to the discrete ones associated with each input neuron was made using the nearest available in the respective range.

Connections between the virtual input layer and the liquid were created using a Gaussian distribution. This curve modulates the values of the weights between pre-synaptic and post-synaptic neurons (only excitatory ones) with a

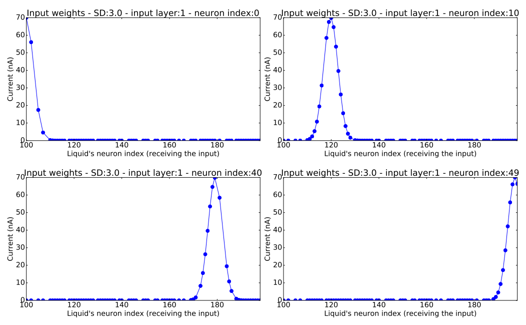

Figure 3. Example of resultant weights values connecting the input to the liquid (standard deviation equals to 3 neuron positions). The abscissae axis has the liquid's neuron index receiving the input spikes.

certain standard deviation (the default value is 3 positions). Therefore, each input connection is spread creating a certain redundancy. Also, in order to increase the separability, each input variable is connected to a unique slice of the liquid.

An example of the resultant weights generated to follow a Gaussian distribution for the second variable (Y-Coordinate of the end position) used in the experiments can be seen in the Fig. 3.

1) Leaky Integrate and Fire: Several spiking neuron models have been proposed in the literature. A comparative study in relation to biological plausibility and computational efficiency was already presented by Izhikevich [29]. As the aim of the current work is to use neuromorphic hardware in the future, the system developed only makes use of the more common Leaky Integrate and Fire neurons (LIF). An LIF neuron was nicely defined by Lewis and Klein [30] as "a capacitor with a decision-making capability".

A variant of the basic LIF model, one with exponential synaptic dynamics, is used in this work. The basic LIF model behaves as a capacitor-resistor circuit with an added circuitry in order to generate the spike (action potential) and also keep it discharged during the refractory period [20]. It can be represented by the set of differential equations (Equations 2, 3 and 4) where c_m is the membrane capacitance (in F), τ_m the membrane time constant (in s), τ_{syn_e} and τ_{syn_i} decay time of the excitatory and inhibitory synaptic current respectively (in s), v_{rest} the membrane resting potential (in V), i_{offset} a fixed noisy current and i_{noise} a variable noisy current (in A). The difference between i_{offset} and i_{noise} is that the first facilitates the creation of a diverse set of neurons (one could think of it as a vector basis) making each neuron react to different inputs and the last is a noise source that could be seen as thermal noise in an electrical circuit. All default values for the variables presented here are available in the Table II.

$$\frac{dv}{dt} = \frac{i_e(t) + i_i(t) + i_{offset} + i_{noise}}{c_m} + \frac{v_{rest} - v}{\tau_m} \tag{2}$$

$$\frac{di_e}{dt} = -\frac{i_e}{\tau_{syn_e}} \tag{3}$$

$$\frac{di_i}{dt} = -\frac{i_i}{\tau_{syn_i}} \tag{4}$$

2) Least squares linear regression: When the LSM framework is employed, usually a linear regression is implemented to train the readout weights connecting the liquid to the output. The generalised linear model (Ordinary Least Square - OLS) from Scikit-learn [31] Python's package was used here. This method solves the following problem:

$$\min_{w} \|Xw - y\|_2{}^2 \tag{5}$$

In this work the matrix X was composed by the membrane low-pass filtered values of the liquid spikes ($\tau_m = 30ms$), y the variable values to be learned and w the readout weights. In order to improve the results, Gaussian White Noise (GWN) with $\mu = 0$ and $\sigma = 0.1$ was added to the X matrix and $\sigma = 0.01$ to the y. As claimed by Bishop [32], the addition of noise can improve the generalisation and could also be seen as analogous to Tikhonov or Ridge Regression.

B. Short-Term Plasticity

In this work, the STP implementation was based on the example from the Brian Simulator documentation (version 1.4) [33], [34]. The Brian's *Synapses* method was used with the parameters from Listing 1 in order to simulate the synaptic dynamics described by Markram et al. [33]. In addition to the implementation described here, the Brian Simulator can also implement the Short-term plasticity using its STP class.

Listing 1. Brian's Synapse STP definition

```
model='''x : 1
         u : 1
         w : 1
         tauf : 1
         taud : 1
         U : 1'''

pre='''u=U+(u-U)*exp(-(t-lastupdate)/tauf)
       x=1+(x-1)*exp(-(t-lastupdate)/taud)
       i+=w*u*x
       x*=(1-u)
       u+=U*(1-u)'''
```

The parameters for the synaptic dynamics are from Maass et al. [6], but with scaling parameter (A) from Joshi and Maass [15]. During the creation of the liquid, all parameters are drawn from a Gaussian distribution whose mean values (μ) are reproduced in Table I and the standard deviation is 50% of $|\mu|$. According to the type of connection (EE, EI, IE and II, where E and I stand for excitatory and inhibitory neuron, respectively) a different value was used. The relation between the parameters from the mentioned papers and the ones from Listing 1 is the following: $A \Rightarrow w$, $U \Rightarrow U$, $D \Rightarrow taud$ and $F \Rightarrow tauf$.

Table I
STP PARAMETERS

Parameter	Value				Unit
	EE	EI	IE	II	
Scaling (A)	70.0	150.0	-47.0	-47.0	nA
τ depression (D)	1.1	0.125	0.7	0.144	ms
τ facilitation (F)	0.05	1.2	0.02	0.06	ms
Use (U)	0.5	0.05	0.25	0.32	-

C. Arm physics simulation

A simulated two-joints planar robot arm (Fig. 1, on the right) was implemented, ignoring friction and gravity. The simulation parameters for the arm have link lengths ($l_1, l_2 = 0.5m$), centre of mass (located in the middle of each link - $l_{c1}, l_{c2} = 0.25m$), moment of inertia ($I_1, I_2 = 0.03kg.m^2$) and mass of $m_1, m_2 = 1.0kg$. The position of the first joint - what in a humanoid robot would be the equivalent to the shoulder - is located on the Cartesian position $(0, 0)$ in order to simplify the simulations. The generated trajectories (Fig. 4) had a total duration of $500ms$ for all experiments always using a time step of $2ms$. Consequently, each trial produced 250 individual values for each variable.

The forward and the inverse kinematics were calculated with a mix of analytical and numerical methods to guarantee always smooth transitions of the joint angles, as it is well known that the inversion of trigonometric functions in most systems produces always answers only in a specific quadrant. Movements were generated (the calculation of the state variables) solving the resultant differential equations of the Lagrangian method [35] in a numerical way using *odeint* from the Python's package SciPy [36]. This method generates the next value based on the time step (Δt) and using the current state variables (position and velocity). From the velocities, the accelerations are calculated and finally the necessary torques to generate the movement. So the system always needs to know the current state and the precision is based on the size of the time step.

D. Generation of the trajectories

A robot arm can only reach positions inside its workspace. The joint ranges were roughly inspired by the ranges from a commercial humanoid robot (Baxter Robot, from Rethink Robotics Inc.). A human arm also has limited joint ranges (try to scratch your back and you will notice them) and this gives support to the design choices made here. In total four trajectories were created (Fig. 4).

The joint angle discretisation was generated creating a linear distribution of values that goes from the lower to the upper joint limit. The result was an array of 50 positions where each position was mapped to a possible floating point value following the input configuration based on a simplified population code (See Section II-A). Array indices behaved as the input neuron index. To find the index yielding the nearest input value, the absolute value of the difference between the input and all the possibles ones available in the array was calculated and the index found by the Numpy method *argmin*. After the visualization of resultant discrete (limited according to the robot's

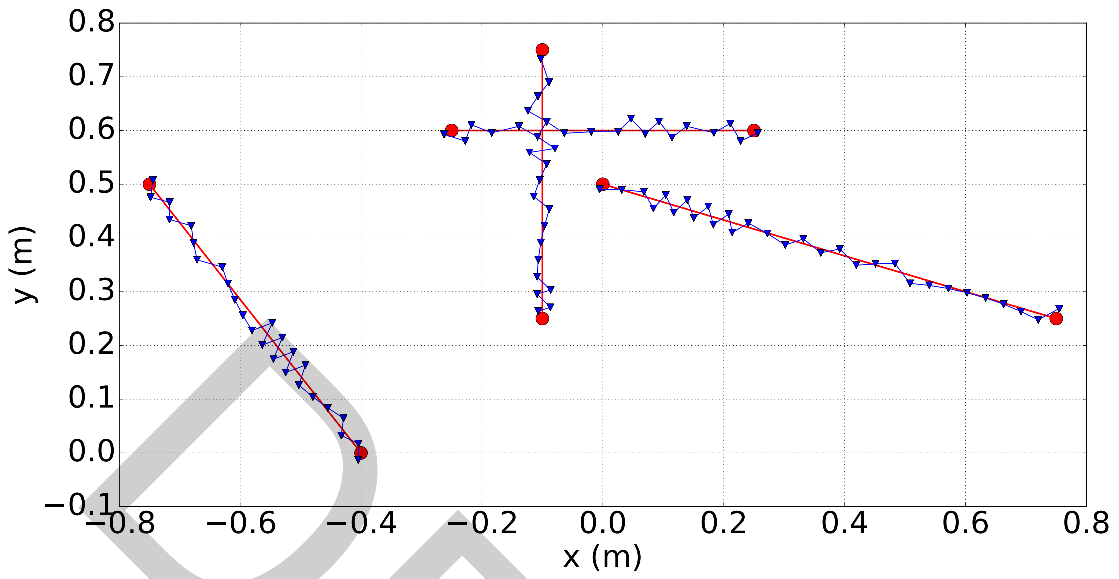

Figure 4. Trajectories used during the experiments. Trajectories 1, 2, 3 and 4 (start-end): $(0.75, 0.25)$-$(0.00, 0.50)$, $(0.25, 0.65)$-$(-0.25, 0.60)$, $(-0.10, 0.75)$-$(-0.10, 0.25)$ and $(-0.75, 0.50)$-$(-0.40, 0.00)$. Discretised versions limited by the arm's workspace are shown with blue triangles.

workspace) and continuous trajectories (Fig. 4), it started to become clear that the generation of the state variables would not work using these discrete trajectories because sharp transitions would generate huge acceleration values.

Trajectory generation starts with endpoint velocity Gaussian shaped curves (calculated as presented by Flash and Hogan [37]) that must be converted to the continuous joint angle space. Using simple derivatives it is possible to calculate the joint velocities and accelerations (not shown here) necessary to generate the torques for the continuous and discrete based trajectories. The calculated torques using the discrete trajectories yielded infeasible values because of abrupt changes in directions. Therefore, the torques generated from the continuous trajectories were used instead because an LSM has a membrane low-pass filter just before the readout layer and that filter acts transforming discrete states into continuous ones. The discretisation is still used when the values are feedback to the LSM though. As an adaptation to increase the input resolution, the torque values were discretised based on the minimum and maximum values generated by the four trajectories during the training phase. Whilst this range selection improves the resolution, it, as well, limits the possible trajectories to be generated and, therefore, the system's generalisation.

E. Brian Simulator

All the SNN simulations used in this work were performed with the Brian simulator [38] version 1.4. Brian is a general purpose system well known by the neuroscience community, freely available, open source and totally based on Python running flawlessly in several different operating systems.

As Brian doesn't have a module specialized for Liquid State Machines, in this work were developed all the tools necessary to automatically generate objects, using the methods available in Brian, in order to simulate the neural network according to what was presented by Maass et al. [6].

1) Brian Step-by-Step extension: In a real-time robot control task, it is mandatory to have a system that can deal with spike trains created as the time goes by. Although Brian is a great software when one needs to simulate off-line systems, the version used here (1.4) cannot easily receive and output a continuous spike train that uses values generated after the simulation had started. To solve this problem, an extension was created to the Brian simulator making possible to simulate on-line systems sending and receiving spike trains.

The extension gives the possibility to inject a virtually infinitely long spike train without running out of memory or slowing down the whole system. At every simulation step, input spikes were sent and output ones received preserving the current simulation state. All the variables used in the simulation became encapsulated into the extension reducing the possibility of bugs. Additionally, the extension properly reinitializes the simulator even when it is used inside the IPython [39] environment and the start-up of the Brian Simulator is performed only once as the extension is initialized, saving time.

F. Experimental set-up

Four sets of experiments (Sets A, B, C and D) were carried out here, all using the same four trajectories (Fig. 4) following a Gaussian endpoint Cartesian velocity profile [37]. Twenty trials were executed to train the readout weights for each trajectory in each set ($20 \times 4 \times 4$).

The simulation's template for all groups was based on the liquid's parameters seen in Table II. Random variables for the generation of the liquid's basic structure (excitatory and inhibitory neurons, connections and weights [6]) were seeded always using the same value (using a Numpy RandomState with seed value equal to 93200) in order to keep the same liquid structure across the simulations. Membrane reset and initial voltages were drawn from a uniform distribution according the limits in Table II. The only parts of the liquid that were not kept the same along the simulations were the variable noisy currents (i_{noise}), the initial membrane voltages and the noisy offset initial currents (i_{offset}) (See Equation 2). The mentioned variables were seeded randomly (Numpy's default) and drawn from its respective distributions for all simulations. In the case of the noise source represented by the variable i_{noise}, a new value was drawn for each simulation's time step.

All experiment sets consisted of two phases: training and testing. During the training phase, a forced teaching approach was used and the data collected was exclusively employed to train the readout weights to generate the trajectories. Because the robot arm model used here has two joints, the system was divided into two readouts - one for each joint - trained individually to generate the next (future) value of the joint's torque based on liquid's current state. Each readout was connected to all 600 liquid neurons through individual membrane low-pass filters and the response from those filters then connected to the readout by 600 individual weights. Those weights were calculated using the Equation 5 during the training phase. During the testing phase the readout weights don't change.

One important point to highlight here is that all trials are always unique. This fact comes from noise injected into the system at every simulation time step in addition to the initialization. The networks presented here are always changing as initialization voltages and offset noises are changed also during the testing. Only the readout weights

Table II
LIQUID DEFAULT PARAMETERS

Parameter	Value	Unit
Membrane time constant (τ_m)	30.0	ms
Membrane capacitance (C_m)	30.0	nF
Synapse time constant (exc. - τ_{syn_e})	3.0	ms
Synapse time constant (inh. - τ_{syn_i})	6.0	ms
Refractory period (exc.)	3.0	ms
Refractory period (inh.)	2.0	ms
Membrane Threshold	15.0	mV
Membrane Reset	[13.8, 14.5]	mV
Membrane Initial	[13.5, 14.9]	mV
i_{offset}	[13.5, 14.5]	nA
i_{noise} (μ)	0.0	nA
i_{noise} (σ)	1.0	nA
Transmission delay (exc.)	1.5	ms
Transmission delay (inh.)	0.8	ms

are kept fixed after training. This is a quite different situation when compared to the conventional way testing is done with artificial neural networks when there's no noise involved, besides the initial one for the training phase.

The training and testing phase experiments were chosen to verify the effects of STP on this specific robotic application.

The noise levels were varied between sets A,B and C,D. According to Rotman et al. [8], STP contributes to information transfer instead of a frequency independent broadband behaviour, therefore noise should be filtered out when STP was active. Also, sets A,B were trained using input noise only at the torques what would make them less robust to feedback errors during the testing phase. All experiments noise levels are specified in relation to the default ones (See Table II). Two types of noisy offset current (see Equation 2) had their values varied, a normally distributed i_{noise} and an uniformly distributed i_{offset}. The i_{noise} default values were $\mu = 0$ and $\sigma = 1nA$ and i_{offset} default ones were from $13.5nA$ to $14.5nA$.

Training Phase: Twelve trials for each trajectory, varying the noise levels and use of STP were executed. The default values for i_{noise} and i_{offset} (see Table II) were varied between sets A,B and C,D. Also the input variables receiving an additive noise were varied between sets A,B and C,D since the system receives feedback from the torques and joint angles. In total, 320 simulations were carried out ($20 \times 4 \times 4$) only for the training of the readouts.

SET A: Using STP, default values and input noise only at the torques.

SET B: Without the use of STP, default values and input noise only at the torques.

SET C: Using STP, i_{noise} and i_{offset} hundred times smaller and input noise at all the input variables.

SET D: Without the use of STP, i_{noise} and i_{offset} hundred times smaller and input noise at all the input variables.

Testing Phase: The trained readouts were tested through a total of 50 new trials for each one of the four trajectories, totalling 800 simulations ($50 \times 4 \times 4$). The results were used as the base to verify the impact of having

STP enabled or not inside the liquid.

G. Analysis tools

1) Dynamic Time Warping: The Dynamic Time Warping (DTW) method [40] generates new pairs that apply a correction (time warping) making it easier to compare two time series (Fig. 5). Using the DTW, the total distance defined by the trajectory formed with the minimum values of the accumulated distance can be easily applied to compare the quality between different trials. The final cost is calculated by summing all the distances (represented by the red lines connecting the both trajectories in Fig. 5). With this metric, the smaller the cost the better the match. The use of this same algorithm as a benchmark for a robotic task was already presented by Azambuja et al. [41]. For the DTW cost calculations in this work we used the algorithm available at github.com/ricardodeazambuja/DTW.

2) Welch's t-test: The Welch's t-test is a variant of the famous Student's t-test. While the last always assumes normally distributed values, equal variances (homoscedasticity) and balanced sample sizes, Welch's was conceived to yield good results when the variances of the two samples are unequal (heteroscedastic) [42]. It is used in this work to verify if two experimental results have equal means (null hypothesis). The Python Scipy package method *ttest_ind* with the option *equal_var=False* was used to calculate the Welch's t-test. According to Chernick and Friis [43], a t-test could still be used when the normality assumption is not satisfied. The justification comes from the central limit theorem because the sample mean will be approximately normal distributed.

The LSMs used here have multiple sources of noise. All the sources are generated using Numpy and could be seen as approximately independent ones. Consequently, according to the central limit theorem, in the limit case, the resultant of the sum of multiple independent noise sources would be normally distributed.

III. Results and Discussion

The results presented in this section come from the four sets of experiments (Section II-F) employed in the control of the simulated robotic arm (Section II-C) aiming to reproduce all the trajectories (Section II-D) shown in Fig. 4. The sets were generated to verify the effects of having STP enabled inside the reservoir. In total 800 simulations were executed to have 50 trials for each trajectory.

The generated trajectories (average values of all 50 trials and its standard errors) are presented in Figs. 6 and 7. Starting by a simple visual inspection, it is clear some trajectories were better executed than others. It is also possible to verify all sets had more problems to reproduce the trajectories 1 and 4 (the inclined ones) when compared to trajectories 2 and 3 (vertical and horizontal straight lines). Fig. 6, the noisier experiments, shows results that are visually very similar (with or without STP), but for the trajectory 2 (triangles). When Fig. 7 is analysed, again the use or not of STP is hard to distinguish, but now the set without STP (Set D) has a better performance for the trajectory 2 (triangles).

The system presented in this paper controls the robot arm generating the torque values necessary to reproduce the trajectories previously learned. It generates the next values by processing the robot arm's proprioceptive feedback.

STP is said to increase the information transmitted [8]. Therefore we would expect the controllers using STP would be more robust to an increase of the noise levels. For this reason, Sets A and B received noise currents one hundred times bigger than sets C and D.

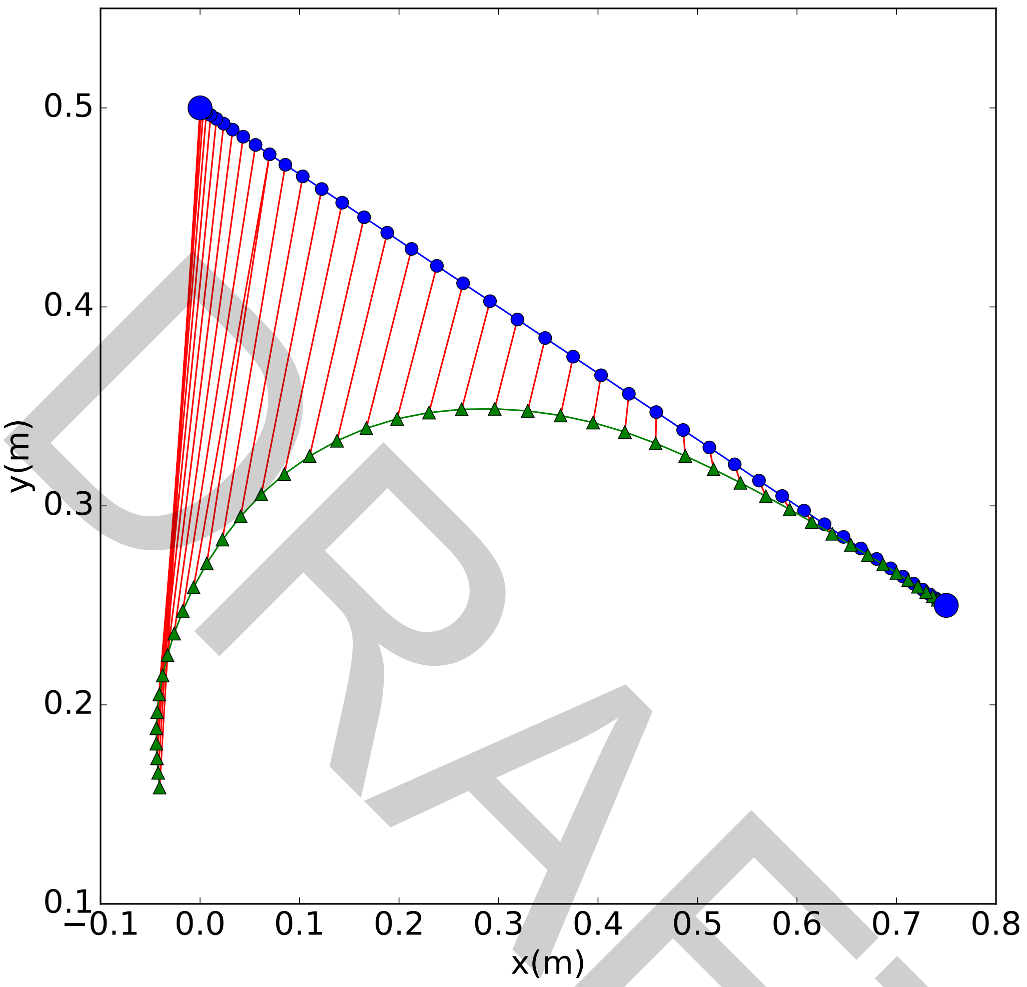

Figure 5. Example of how the DTW compares one subsampled trajectory generated by the LSM (green triangles) against the original one (blue circles).

Looking at the amount of spikes generated during all the trials (Figs 8 and 9), the oscillations inside the liquid have almost the same period when comparing the effects of STP. Still for the noisy sets (A and B) only during the first 25 steps the reservoir using STP generated fewer spikes, while for the sets C and D the number of spikes was always smaller with STP. This could suggest STP was filtering out useful spikes instead of noise in view of the fact in Fig. 8, which has the output from the noisier sets (A and B), there's almost no difference between the number of spikes with or without STP.

In the Fig. 10 it is shown raster plots for all sets during one trial while reproducing the trajectory 1. The sets

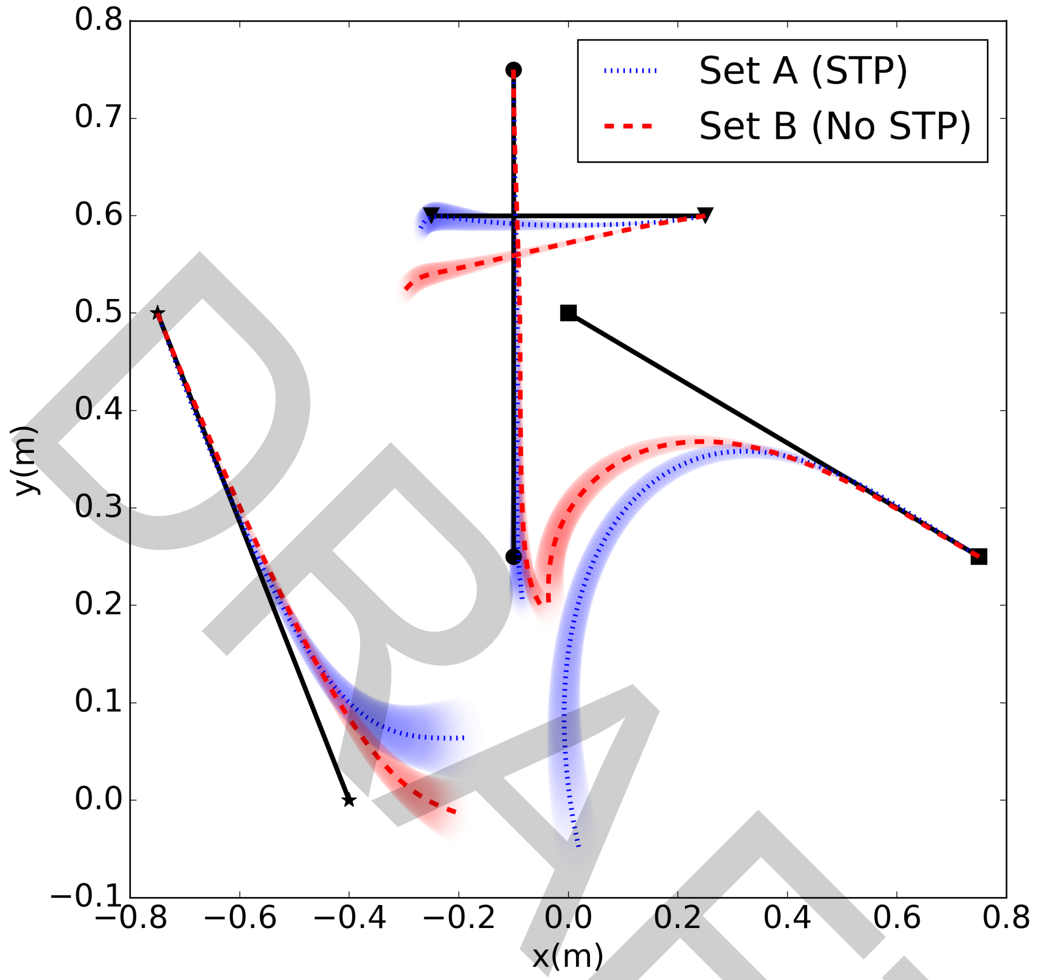

Figure 6. Resultant trajectories for all fifty trials - average values for Sets A and B. Standard error is represented by the shaded areas.

using STP (A and C) take longer to fill the vertical axis with spikes when compared to the ones without STP (B and C). This confirms the averaged values from Figs. 8 and 9.

Besides being visually appealing, the averaged trajectories (Figs. 6 and 7) could misrepresent the results since they have problems to deal with time distortions [41]. Using DTW, it is possible to correct, up to a certain extension, those problems resultant from the simple averaging of the trial results. Therefore, concerning the performance of STP, the DTW costs (see Section II-G1) were calculated for each individual trajectory (Fig.11). The statistical verification of the results was made using Welch's t-test (see Section II-G2).

The results from the Welch's t-test (See Fig. 11) shown the sets using STP (A and C) performed better only

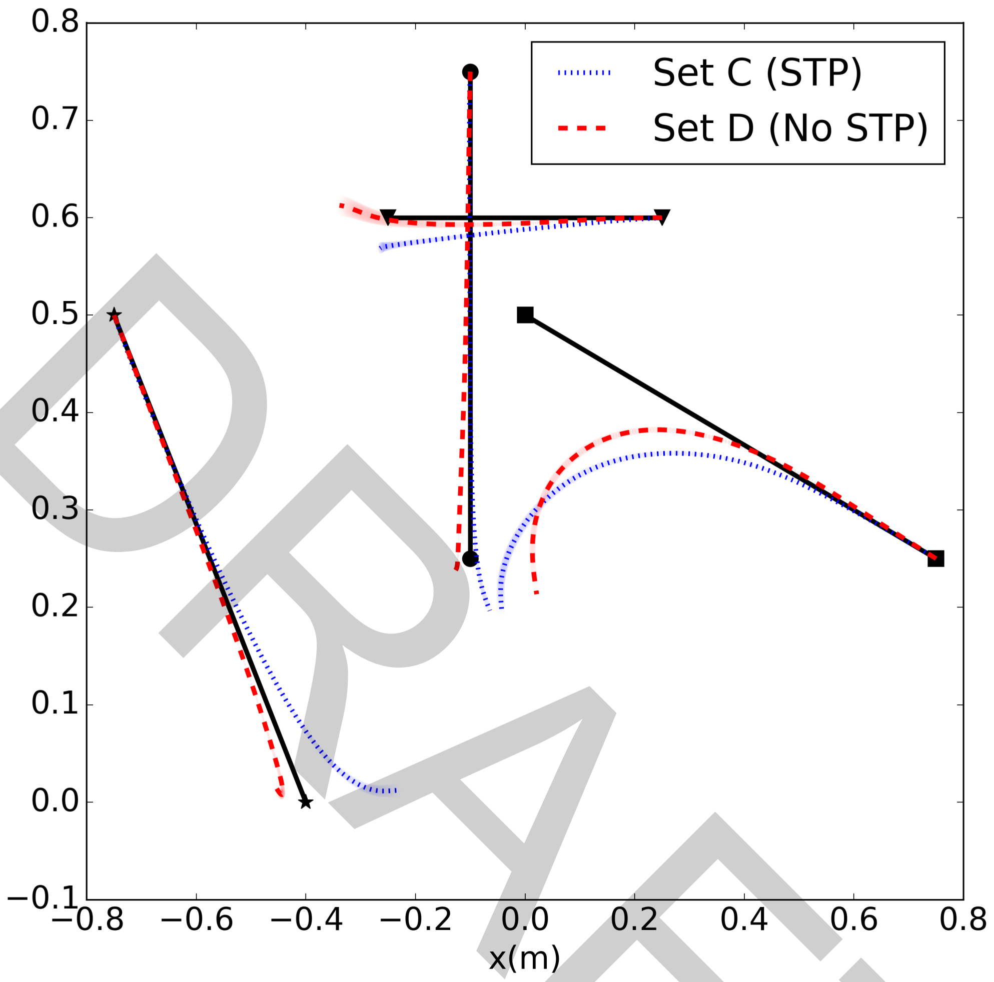

Figure 7. Resultant trajectories for all fifty trials - average values for Sets C and D. Standard error is represented by the shaded areas.

during the trajectory 2 (triangles). For the trajectory 3, the results were not distinguishable by the statistical tests since *p-values* were bigger than 0.05. The trajectory 4 had, again, no differences for the noisy sets (A and B). However, the set D performed better than C here too. Finally, results for trajectory 1 were always better when no STP was employed.

IV. Conclusions and Future Works

In an SNN simulation, STP is a computationally very expensive factor and, consequently, it is important to verify if the system could work as well as without it in a robotic controller implementation like the one presented here. Neuromorphic systems as SpiNNaker [10] can have the maximum number of neurons simulated greatly reduced

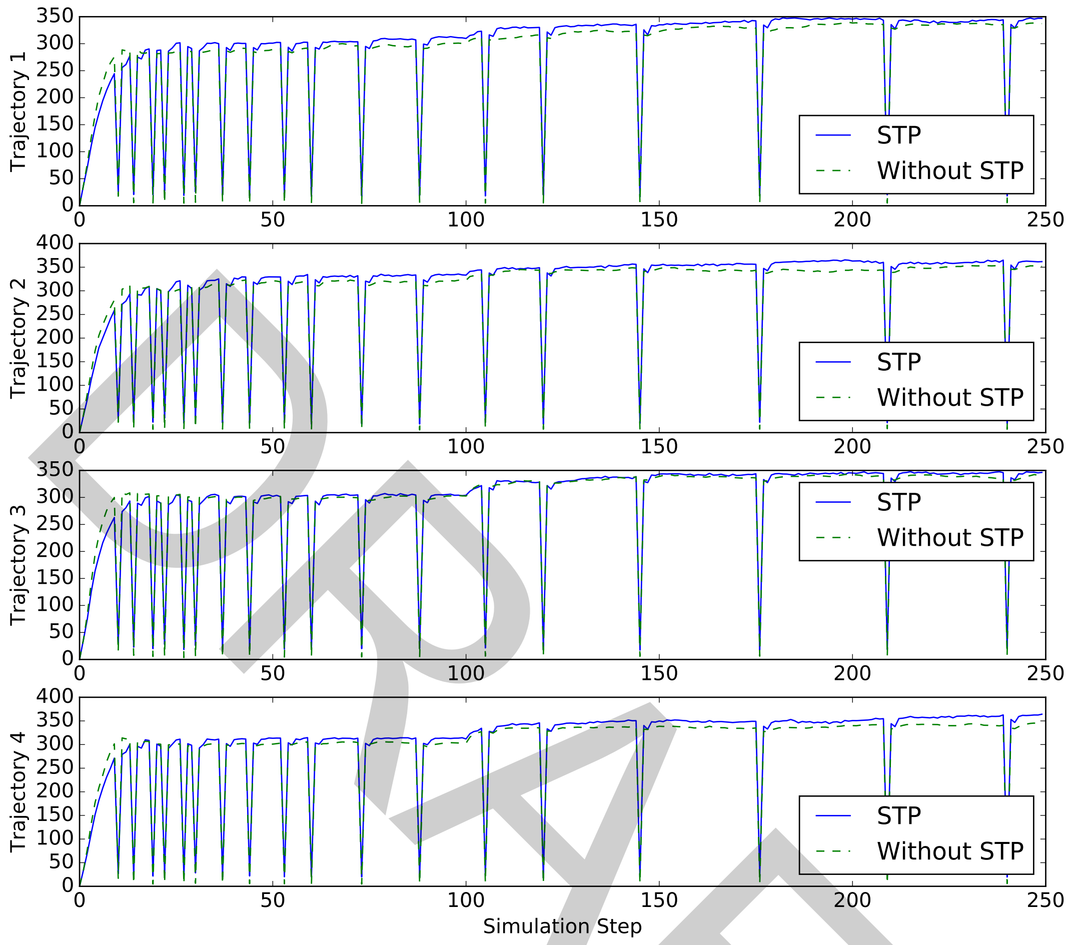

Figure 8. Average number of emitted spikes 50 trials for Sets A (dashed lines) and B (continuous lines) with and without STP, respectively.

when extra rules as the one closely related to STP (up-to-date, it has not been implemented in SpiNNaker), the Spike-Timing Dependent Plasticity (STDP), are used.

The literature states STP can be important for the generation of motor commands [44] and together with dynamic firing threshold could improve the performance of an LSM during a classification task of a synthetic data set [45].

From our results, the application of STP inside the reservoir had clear beneficial effects only for one out of the three tested trajectories. Therefore, considering the computational costs involved when STP is used, it will be necessary more studies to really confirm the advantages of STP in this type of robotic application and it will be let as future works.

The implementation details and source code for this work can be found on github.com/ricardodeazambuja/IJCNN2017.

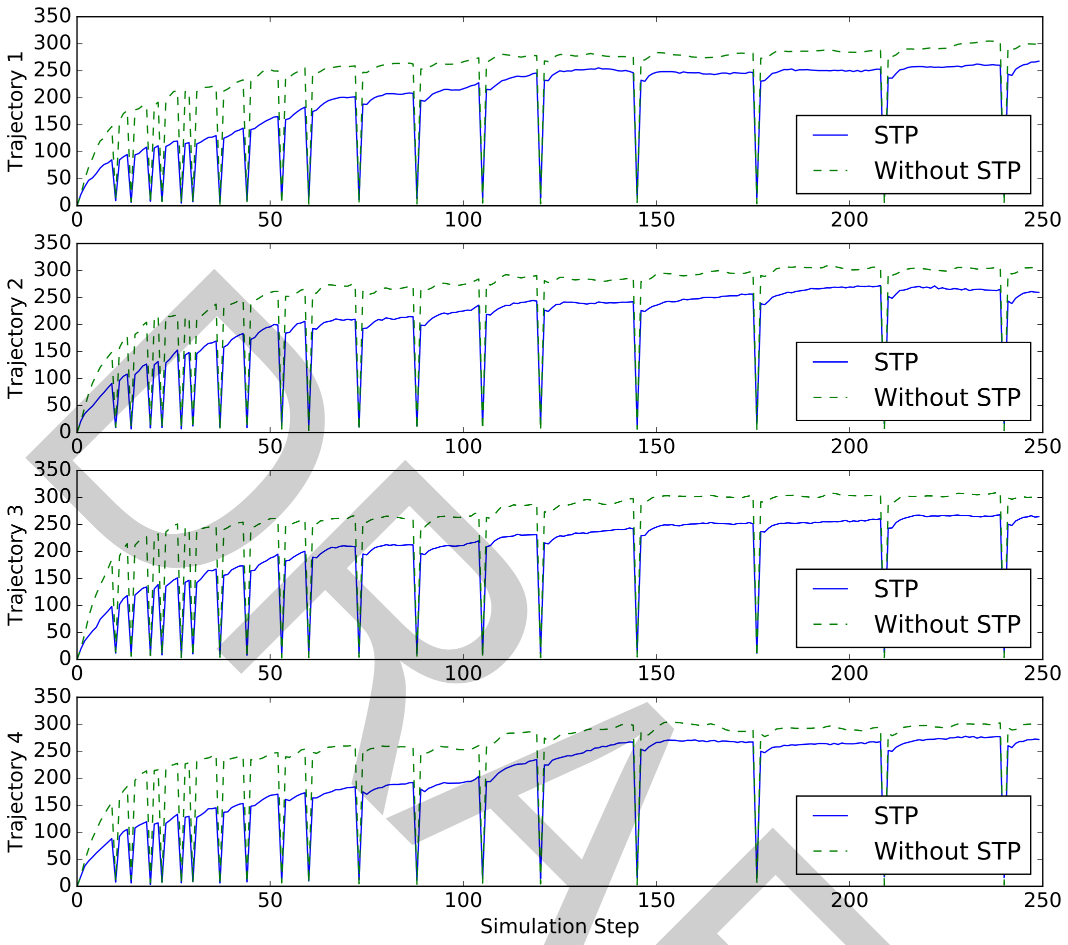

Figure 9. Average number of emitted spikes 50 trials for Sets C (dashed lines) and D (continuous lines) with and without STP, respectively).

Acknowledgment

This work was in part supported by the CAPES Foundation, Ministry of Education of Brazil (scholarship BEX 1084/13-5), CNPq Brazil (scholarship 232590/2014-1) and UK EPSRC project BABEL (EP/J004561/1 and EP/J00457X/1).

References

[1] J. G. Taylor, "Cognitive Computation," *Cognitive Computation*, vol. 1, no. 1, pp. 4–16, Mar. 2009.

[2] W. Maass, "Networks of Spiking Neurons: The Third Generation of Neural Network Models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, Dec. 1997.

[3] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015.

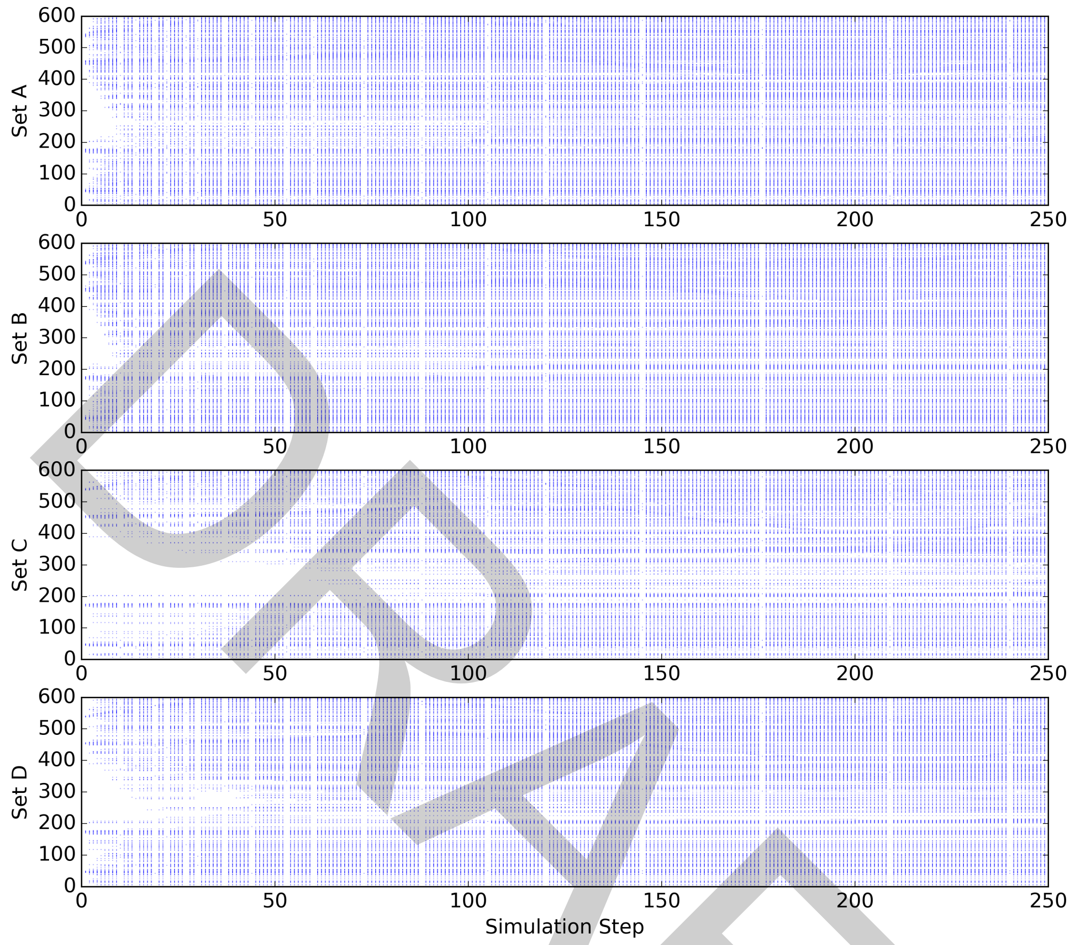

Figure 10. An example of the spikes generated inside liquid when reproducing the Trajectory 1 (See Fig. 4).

[4] S. Thorpe, A. Delorme, and R. Van Rullen, "Spike-Based Strategies for Rapid Processing," *Neural networks*, vol. 14, no. 6, pp. 715–725, 2001.

[5] S. V. Adams, A. D. Rast, C. Patterson, F. Galluppi, K. Brohan, J.-A. Pérez-Carrasco, T. Wennekers, S. Furber, and A. Cangelosi, "Towards Real-World Neurorobotics: Integrated Neuromorphic Visual Attention," in *Neural Information Processing*. Springer, 2014, pp. 563–570.

[6] W. Maass, T. Natschläger, and H. Markram, "Real-Time Computing without Stable States: A New Framework for Neural Computation Based on Perturbations," *Neural computation*, vol. 14, no. 11, pp. 2531–2560, Nov. 2002.

[7] W. Maass and H. Markram, "Synapses as Dynamic Memory Buffers," *Neural Networks*, vol. 15, no. 2, pp. 155 – 161, 2002.

[8] Z. Rotman, P.-Y. Deng, and V. A. Klyachko, "Short-Term Plasticity Optimizes Synaptic Information Transmission," *Journal of Neuroscience*, vol. 31, no. 41, pp. 14 800–14 809, Oct. 2011.

[9] W. Maass, T. Natschläger, and H. Markram, "Fading Memory and Kernel Properties of Generic Cortical Microcircuit Models," *Journal of Physiology-Paris*, vol. 98, no. 4–6, pp. 315–330, Jul. 2004.

[10] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the SpiNNaker System Architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, Dec. 2013.

[11] J. s Seo, B. Brezzo, Y. Liu, B. D. Parker, S. K. Esser, R. K. Montoye, B. Rajendran, J. A. Tierno, L. Chang, D. S. Modha, and D. J. Friedman, "A 45nm CMOS Neuromorphic Chip with a Scalable Architecture for Learning in Networks of Spiking Neurons," in *2011 IEEE Custom Integrated Circuits Conference (CICC)*, Sep. 2011, pp. 1–4.

[12] B. V. Benjamin, Peiran Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.

[13] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, "A Wafer-Scale Neuromorphic Hardware System for Large-Scale Neural Modeling," *Proceedings of the 2010 IEEE International Symposium on Circuits and Systems (ISCAS"10)*, pp. 1947–1950, 2010.

[14] M. Noack, J. Partzsch, C. G. Mayr, S. Hänzsche, S. Scholze, S. Höppner, G. Ellguth, and R. Schüffny, "Switched-Capacitor Realization of Presynaptic Short-Term-Plasticity and Stop-Learning Synapses in 28 Nm CMOS," *Frontiers in Neuroscience*, vol. 9, Feb. 2015.

[15] P. Joshi and W. Maass, "Movement Generation with Circuits of Spiking Neurons," *Neural Computation*, vol. 17, no. 8, pp. 1715–1738, 2005.

[16] Y. Zhang, P. Li, Y. Jin, and Y. Choe, "A Digital Liquid State Machine With Biologically Inspired Learning and Its Application to Speech Recognition," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–1, 2015.

[17] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout, "Isolated Word Recognition with the Liquid State Machine: A Case Study," *Information Processing Letters*, vol. 95, no. 6, pp. 521–528, Sep. 2005.

[18] H. Burgsteiner, M. Kröll, A. Leopold, and G. Steinbauer, "Movement Prediction from Real-World Images Using a Liquid State Machine," in *Innovations in Applied Artificial Intelligence*. Springer, 2005, pp. 121–130.

[19] F. Schürmann, K. Meier, and J. Schemmel, "Edge of Chaos Computation in Mixed-Mode Vlsi-a Hard Liquid," in *Advances in Neural Information Processing Systems*, 2004, pp. 1201–1208.

[20] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge university press, 2002.

[21] D. S. Bassett and E. Bullmore, "Small-World Brain Networks," *The Neuroscientist*, vol. 12, no. 6, pp. 512–523, Dec. 2006.

[22] D. J. Watts and S. H. Strogatz, "Collective Dynamics of 'small-World' Networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[23] D. V. Buonomano and W. Maass, "State-Dependent Computations: Spatiotemporal Processing in Cortical Networks," *Nature Reviews Neuroscience*, vol. 10, no. 2, pp. 113–125, Feb. 2009.

[24] E. T. Rolls and G. Deco, *The Noisy Brain: Stochastic Dynamics as a Principle of Brain Function*. Oxford university press New York, 2010, vol. 28.

[25] M. D. McDonnell and L. M. Ward, "The Benefits of Noise in Neural Systems: Bridging Theory and Experiment," *Nature Reviews Neuroscience*, vol. 12, no. 7, pp. 415–426, 2011.

[26] K. P. Dockendorf, I. Park, P. He, J. C. Príncipe, and T. B. DeMarse, "Liquid State Machines and Cultured Cortical Networks: The Separation Property," *Biosystems*, vol. 95, no. 2, pp. 90–97, Feb. 2009.

[27] W. Maass, P. Joshi, and E. D. Sontag, "Computational Aspects of Feedback in Neural Circuits," *PLoS Comput Biol*, vol. 3, no. 1, p. e165, Jan. 2007.

[28] H. Hauser, R. M. Füchslin, and R. Pfeifer, *Opinions and Outlooks on Morphological Computation*, H. Hauser, R. M. Füchslin, and R. Pfeifer, Eds., Zürich, 2014.

[29] E. Izhikevich, "Which Model to Use for Cortical Spiking Neurons?" *IEEE Transactions on Neural Networks*, vol. 15, no. 5, pp. 1063–1070, Sep. 2004.

[30] M. A. Lewis and T. J. Klein, "Neurorobotics Primer," in *The Path to Autonomous Robots*. Springer, 2009, pp. 1–25.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-Learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[32] C. M. Bishop, "Training with Noise Is Equivalent to Tikhonov Regularization," *Neural Computation*, vol. 7, no. 1, pp. 108–116, Jan. 1995.

[33] H. Markram, Y. Wang, and M. Tsodyks, "Differential Signaling via the Same Axon of Neocortical Pyramidal Neurons," *Proceedings of the National Academy of Sciences*, vol. 95, no. 9, pp. 5323–5328, 1998.

[34] G. Mongillo, O. Barak, and M. Tsodyks, "Synaptic Theory of Working Memory," *Science*, vol. 319, no. 5869, pp. 1543–1546, Mar. 2008.

[35] P. L. Gribble and D. J. Ostry, "Compensation for Interaction Torques during Single-and Multijoint Limb Movement," *Journal of neurophysiology*, vol. 82, no. 5, pp. 2310–2326, 1999.

[36] E. Jones, T. Oliphant, P. Peterson, and others, *SciPy: Open Source Scientific Tools for Python*, 2001–, [Online; accessed 2015-06-02].

[37] T. Flash and N. Hogan, "The Coordination of Arm Movements: An Experimentally Confirmed Mathematical Model," *The journal of Neuroscience*, vol. 5, no. 7, pp. 1688–1703, 1985.

[38] D. F. M. Goodman, "The Brian Simulator," *Frontiers in Neuroscience*, vol. 3, no. 2, pp. 192–197, Sep. 2009.

[39] F. Pérez and B. E. Granger, "IPython: A System for Interactive Scientific Computing," *Computing in Science and Engineering*, vol. 9, no. 3, pp. 21–29, May 2007.

[40] H. Sakoe and S. Chiba, "Dynamic Programming Algorithm Optimization for Spoken Word Recognition," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 26, no. 1, pp. 43–49, Feb. 1978.

[41] R. de Azambuja, A. Cangelosi, and S. Adams, "Diverse, Noisy and Parallel: A New Spiking Neural Network Approach for Humanoid Robot Control," in *2016 International Joint Conference on Neural Networks (IJCNN)*, Vancouver, July 24-29 2016, pp. 1134–1142.

[42] G. D. Ruxton, "The Unequal Variance T-Test Is an Underused Alternative to Student's T-Test and the Mann–Whitney U Test," *Behavioral Ecology*, vol. 17, no. 4, pp. 688–690, Jan. 2006.

[43] M. R. Chernick and R. H. Friis, *Introductory Biostatistics for the Health Sciences: Modern Applications Including Bootstrap*, 1st ed. Hoboken, N.J: Wiley-Interscience, Mar. 2003.

[44] L. Martin, B. Sándor, and C. Gros, "Closed-Loop Robots Driven by Short-Term Synaptic Plasticity: Emergent Explorative vs. Limit-Cycle Locomotion," *arXiv preprint arXiv:1608.02838*, 2016.

[45] S. Schliebs, M. Fiasché, and N. Kasabov, "Constructing Robust Liquid State Machines to Process Highly Variable Data Streams," in *Artificial Neural Networks and Machine Learning–ICANN 2012*. Springer, 2012, pp. 604–611.

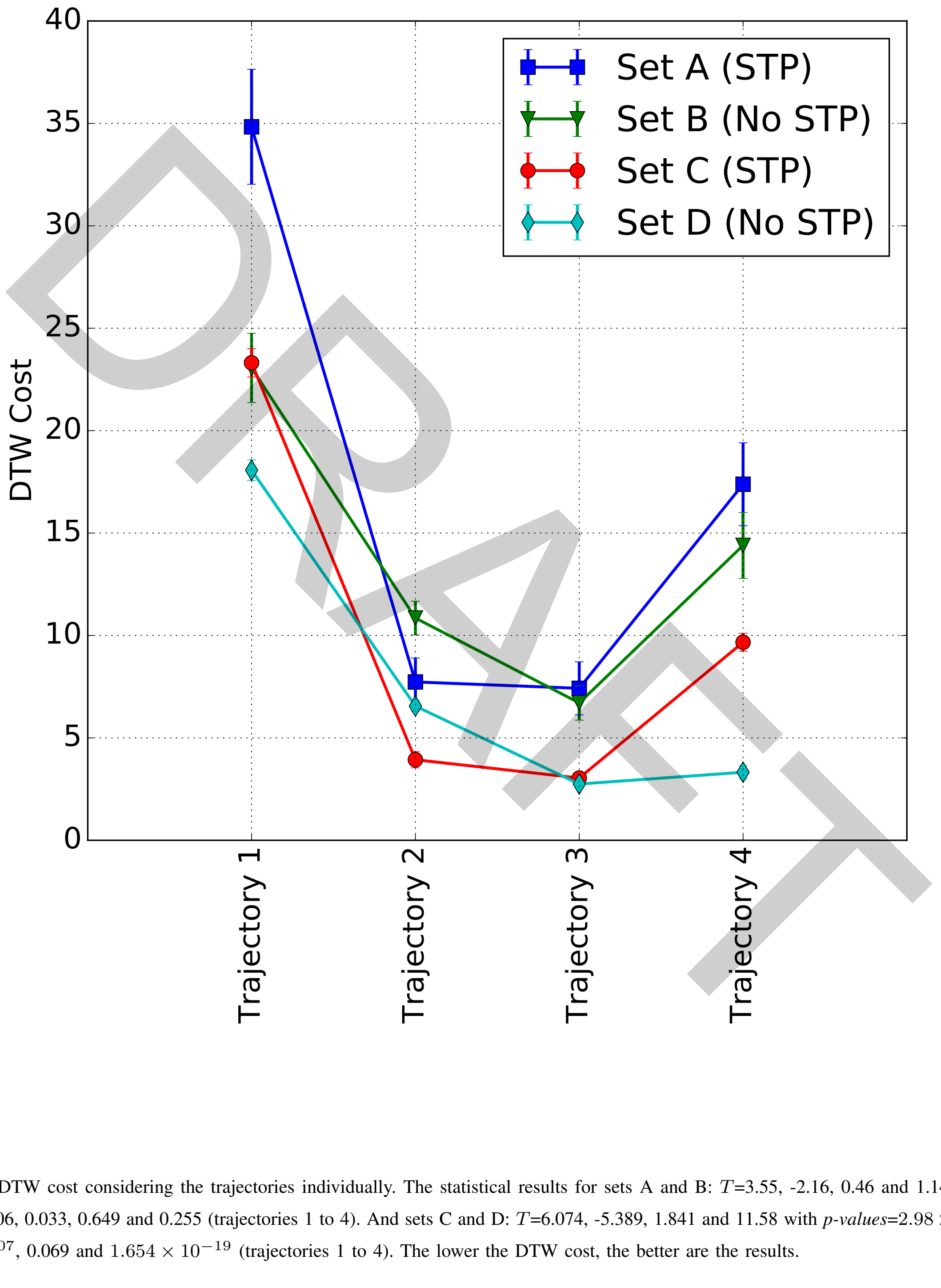

Figure 11. DTW cost considering the trajectories individually. The statistical results for sets A and B: T=3.55, -2.16, 0.46 and 1.14 with *p-values*=0.0006, 0.033, 0.649 and 0.255 (trajectories 1 to 4). And sets C and D: T=6.074, -5.389, 1.841 and 11.58 with *p-values*=2.98×10^{-08}, 5.96×10^{-07}, 0.069 and 1.654×10^{-19} (trajectories 1 to 4). The lower the DTW cost, the better are the results.