Training a Spiking Neural Network to Control a 4-DoF Robotic Arm based on Spike Timing-Dependent Plasticity

Alexandros Bouganis and Murray Shanahan

***Abstract*— In this paper, we present a spiking neural network architecture that autonomously learns to control a 4 degree-of-freedom robotic arm after an initial period of motor babbling. Its aim is to provide the joint commands that will move the end-effector in a desired spatial direction, given the joint configuration of the arm. The spiking neurons have been simulated according to Izhikevich's model, which exhibits biologically realistic behaviour and yet is computationally efficient. The architecture is a feed-forward network where the input layers encode the intended movement direction of the end-effector in spatial coordinates, as well as the information that is given by proprioception about the current joint angles of the arm. The motor commands are determined by decoding the firing patterns in the output layers. Both excitatory and inhibitory synapses connect the input and output layers, and their initial weights are set to random values. The network learns to map input stimuli to motor commands during a phase of repetitive action-perception cycles, in which Spike Timing-Dependent Plasticity (*STDP*) strengthens synapses between neurons that are correlated and weakens synapses between uncorrelated ones. The trained spiking neural network has been successfully tested on a kinematic model of the arm of an iCub humanoid robot.**

I. Introduction

In this work, we present a neural network architecture that autonomously learns to control a four degree-of-freedom robotic arm in the three dimensional space. The problem of controlling a robotic arm has attracted the attention of many researchers in the past, with a common assumption being that the kinematic model of the arm is *a priori* known. This assumption enabled researchers to either introduce analytical methods, which offer exact solutions for simple kinematic chains, or propose solutions based on numerical methods. Recently, there has been increasing interest in developing methods that do not assume any *a priori* knowledge for the arm's kinematic model, but its kinematic properties are derived through a learning procedure.

Bullock *et al.* [1] presented the DIRECT (Direction-to-Rotation Effector Control Transform) model which is a self-organizing neural network that learns, during a motor babbling period, the mapping between joint commands and the resulting spatial displacements of the end-effector. (Motor babbling can be observed in babies, where a repetitive action-perception cycle generates associative information between the various representations.) Action is generated through the Endogenous Random Generator (*ERG*) [2], which sends random motor commands, and the results of these actions in the spatial domain are perceived and associated. Outstar-learning [3] is used in the DIRECT model for training the network and modifying the synaptic weights. More recently, Asuni *et al.* [4] were inspired by this approach and proposed a neural network, also based on outstar learning, that aims to control a robotic head for gazing points in the 3D space.

Alexandros Bouganis and Murray Shanahan are with the Department of Computing, Imperial College London, UK (email: {alexandros.bouganis, m.shanahan}@imperial.ac.uk).

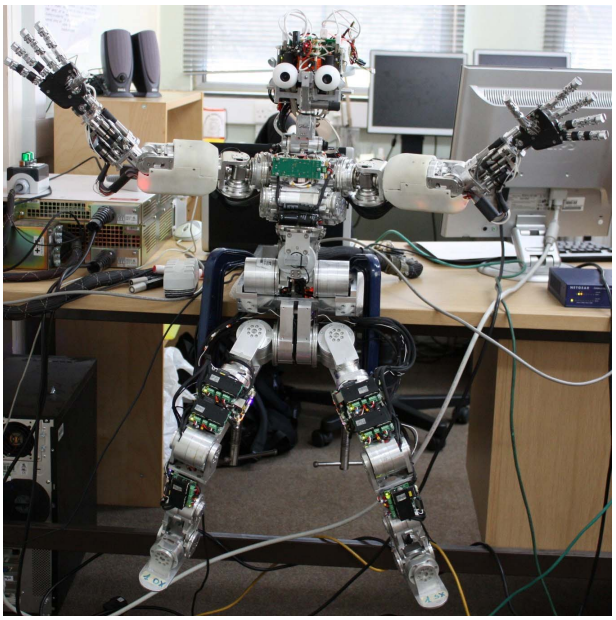

Fig. 1. The iCub.

The present work is also influenced by the DIRECT model. An important feature of the proposed network is that it consists of individual *spiking neurons* that exhibit realistic behaviour and uses a biologically plausible learning mechanism for modifying the synaptic weights, namely Spike Timing-Dependent Plasticity (STDP). Spiking neural networks are considered to be biologically realistic and many researchers have lately used them for coordinate transformations ([5], [6]), object segmentation [7], visual pattern recognition [8], *etc*.

The present paper is organized as follows. Section II presents background information on controlling a robotic arm. Section III introduces the spiking neural network, the model of the spiking neuron, and the STDP learning mechanism. Experiments are presented in section IV, while the conclusions are given in section V.

978-1-4244-8126-2/10/$26.00

II. Reaching With A Robotic Arm

Many tasks in robotics require the robot's end-effector to move between two points in space. The issue that arises, however, is that while the task is naturally represented in cartesian coordinates (world coordinates), the robot is only able to control its arm through the motors and perform the task in the joint space. The computation of the joint coordinates that result in a desired spatial position of the end effector is called inverse kinematics, and the problem is ill-posed when considering arms with redundant degrees of freedom since there could be multiple solutions. A simple way to move the end-effector to its target position can be achieved by considering each joint independently, and incrementing its angle accordingly towards its final value at the target configuration of the arm. However, this approach is not competent for most applications as it cannot force the end-effector to follow a desired path between the initial and the target position. A synergy of angle changes at joints is required. This issue can be resolved though by taking many intermediate points along the desired path of the end-effector, and using these points as "mid-term targets". The sequence of the joint angles that fix the end-effector at the intermediate points can then be followed, forcing the end-effector to trace the desired path. As discussed in [1], this approach has two main shortcomings. Since there could be multiple joint configurations that can result to the same spatial position of the end-effector, it is possible to have discontinuity in the joint space, and encounter, for example, two non-adjacent angles between consecutive steps along the path for the same joint. Moreover, due to the non-linearity of the mapping function between joint angles and spatial position of the end-effector, the linear combination of solutions is probably not a solution itself.

To overcome these issues, we can follow an alternative approach, and instead of computing the *joint angles* that result in the desired *spatial position* of the end effector, we can use small steps towards the target position and compute at each step the *joint velocities* that move the end-effector in the desired *spatial direction*. More specifically, let us assume that the manipulator has M degrees of freedom and the joint angles are denoted by $\boldsymbol{\vartheta} = [\vartheta_1, \vartheta_2, \ldots, \vartheta_M]$. Let us also assume that the spatial position of the end-effector is represented by the N-dimensional vector $\mathbf{e} = [e_1, e_2, \ldots, e_N]$. If we assume that the current joint configuration is given by $\boldsymbol{\vartheta}$, and the joint velocities $\dot{\boldsymbol{\vartheta}}$ result in the end-effector's spatial velocity $\dot{\mathbf{e}}$, then:

$$\dot{\mathbf{e}} = \mathbf{J}(\boldsymbol{\vartheta}) \cdot \dot{\boldsymbol{\vartheta}} \quad (1)$$

$$\mathbf{J}(\boldsymbol{\vartheta}) = \begin{bmatrix} \frac{\partial e_1}{\partial \vartheta_1} & \frac{\partial e_1}{\partial \vartheta_2} & \cdots & \frac{\partial e_1}{\partial \vartheta_M} \\ \frac{\partial e_2}{\partial \vartheta_1} & \frac{\partial e_2}{\partial \vartheta_2} & \cdots & \frac{\partial e_2}{\partial \vartheta_M} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial e_N}{\partial \vartheta_1} & \frac{\partial e_N}{\partial \vartheta_2} & \cdots & \frac{\partial e_N}{\partial \vartheta_M} \end{bmatrix} \quad (2)$$

where $\mathbf{J}$ denotes the Jacobian and its value depends on the joint configuration of the arm. As can be seen, each column of the Jacobian corresponds to a single joint of the robot and indicates the effect that a small change in its angle has on the spatial position of the end-effector. The change in the spatial position of the end-effector can be computed by the sum of the individual changes in its position by each one degree of freedom. Given that our aim is to compute the joint commands $\dot{\boldsymbol{\vartheta}}$ that will result in moving the end-effector in the desired spatial direction $\dot{\mathbf{e}}$, we can use eq.(1) and solve with respect to $\dot{\boldsymbol{\vartheta}}$:

$$\dot{\boldsymbol{\vartheta}} = \mathbf{J}^{\sharp}(\boldsymbol{\vartheta}) \cdot \dot{\mathbf{e}} \quad (3)$$

$$\mathbf{J}^{\sharp}(\boldsymbol{\vartheta}) = \left(\mathbf{J}^T\mathbf{J}\right)^{-1}\mathbf{J}^T \quad (4)$$

where $\mathbf{J}^{\sharp}$ and $\mathbf{J}^T$ denote respectively the pseudo-inverse and the transpose of the Jacobian at the joint configuration $\boldsymbol{\vartheta}$.

Thus, the approach is to (i) decompose the path that the end-effector should follow into small steps, (ii) compute at each step the spatial direction for the next movement, (iii) compute the Jacobian at the current joint configuration (since its value is only valid locally in the joint space), (iv) compute the pseudo-inverse of the Jacobian, and finally, (v) compute the joint commands $\dot{\boldsymbol{\vartheta}}$ according to eq.(3). The linearity of eq.(1) ensures that the linear combination of known solutions will also give a valid solution. Also, the fact that the solution found emerges by computing the small increments in the joint angles ensures the continuity in the joint space along the path.

In this paper, we present a spiking neural network that exhibits the same functionality as the approach described above, and consists of spiking neurons that have been modeled by Izhikevich's equations [10]. This neuron model is chosen as it facilitates efficient simulation of real neurons with biologically realistic behaviours. The network autonomously learns to control a robotic arm with four degrees-of-freedom in 3D space (we are interested only in the position, and not orientation, of the end-effector) during an initial period of motor babbling, using STDP ([11], [17]) to strengthen or weaken the synaptic weights between sensory and motor neurons which are originally randomly set. During the period of the action-perception cycle, proprioception stimulates sensory neurons and encodes into the network the current joint configuration. The Endogenous Random Generator (*ERG*) randomly stimulates motor neurons and the resulting joint commands move the end-effector in a certain spatial direction which is observed and encoded into the network. The temporal correlation between neuronal firings is extracted by the Spike Timing-Dependent Plasticity mechanism which modifies the synaptic weights so that if the robotic arm is at the pose just learned and the end-effector should move in the same spatial direction, then the motor neurons that were originally activated by ERG, should now be stimulated by the current they receive from the input neurons. Currently, the spatial position of the end-effector in the training stage is computed by solving forward kinematic equations. Future work will substitute this by including a visual pathway in the network. Further details about the network and the learning mechanism are given in the following section.

III. The Spiking Neural Network

The proposed neural network consists of spiking neurons which are organized into seven input layers and four output layers, as shown in Figure 2.[1] We will denote for the rest of the paper the i-th input layer and the j-th output layer by L_i^{input} and L_j^{output}, respectively. We use a population of 1200 neurons for each input layer, and a population of 800 neurons for each output layer. Four of the input layers $L_{i=1:4}^{input}$ encode the information that is given by proprioception, and the firing pattern at each one of them indicates the angle at the respective joint. The four joints of interest are located at the shoulder (roll, pitch and yaw) and the elbow of the arm, with their ranges being $[-75^o, -15^o]$, $[15^o, 75^o]$, $[-10^o, 50^o]$ and $[15^o, 75^o]$. The network encodes these angles after discretizing them into bins with 5^o resolution. The remaining three input layers $L_{i=5:7}^{input}$ represent the spatial direction that the end-effector should move at the next time step, with each layer encoding the projection of the 3D directional vector to one of the world axes. The directions encoded are also discretized using a 45^o resolution, resulting in 26 possible movements of the end-effector from its original position. The input layers are connected all-to-all with the output layers, with the firing pattern of each output layer representing the motor command that is provided to a single joint. Each neuron in the input layers is connected with an excitatory and inhibitory synapse to each output neuron. This is equivalent to representing a single input neuron by a pair of two highly correlated neurons, one excitatory and one inhibitory. All synapses are plastic, which means that their original random weights are modified during the motor babbling period under STDP.

A common assumption made is that neuronal activity patterns represent a single value per variable at any given time [12]. Biological evidence also supports that the activity level of a population of neurons is characterized by tuning curves, typically bell-shaped, which describe the mean firing rates of neurons based on the value of the represented variable. The peak of that curve indicates the "central neuron" which exhibits the highest sensitivity for a given value of the variable [13]. In this work, a Gaussian distribution is used to model the tuning curves in the neuronal layers. Let us assume that a population of n neurons represents a variable θ, whose domain is $[0, 1]$ after normalization. If $f : [0, 1] \rightarrow [1, n]$ indicates the neuron that exhibits the highest activity for a specific value of the variable, then the distribution of the firing rate is given by:

$$F(x) = F_{max} \cdot e^{-\frac{(x-f(\theta_0))^2}{2\sigma^2}} \tag{5}$$

where σ denotes the standard deviation, F_{max} is the maximum firing rate, and $F(x)$ expresses the firing rate of neuron x when the normalized value θ_0 is encoded. While the process of encoding the value of a variable into spike trains is important for the input layers, as well as for the output layers in the training phase, the reverse process of decoding firing patterns from the output layers to motor commands is important in the "performance period". A voting scheme is adopted [12], according to which the "central neuron" is given by:

$$r = \frac{\sum_{x=1}^{n} F(x)x}{\sum_{x=1}^{n} F(x)} \tag{6}$$

while the normalized motor command is given by $f^{-1}(r)$.

We should now discuss two issues that arise due to the representation scheme adopted and the nature of the task. As has been described, we have seven independent layers of neurons which represent the joint configuration of the arm and the intended spatial direction of movement of the end-effector, as well as four layers of motor neurons that control the four joints. In the task of controlling the robotic arm, it is highly likely to encounter two joint configurations $\boldsymbol{\vartheta}_1$ and $\boldsymbol{\vartheta}_2$ that differ only in the angle of a single joint, and the movement of the end-effector in the same spatial direction at the two configurations requires two different sets of motor commands. In terms of the spiking neural network this entails that the stimulation in the input layers of two neuronal populations, which largely overlap and differ only by a small subset of firing neurons, should be able to result in the activation of two different sets of motor neurons. This is autonomously accomplished by the proposed network, which includes all-to-all connections between the input and the output neurons, and modifies the initial synaptic weights through the STDP learning mechanism. STDP manages to strengthen (weaken) the inhibitory (respectively, excitatory) synapses between uncorrelated sensory and motor neurons, and have the opposite effect on synapses between correlated neurons. A balance is also important between the maximum allowable inhibitory and excitatory synaptic weights with respect to the minimum number of firing neurons that can differ between two potential input patterns. To see the second issue that arises, let us assume that during the motor babbling

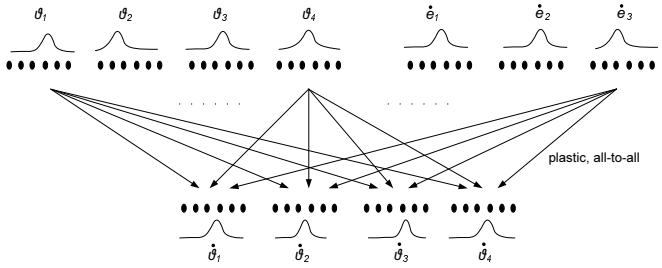

Fig. 2. Architecture of the feed-forward network. The network includes seven input layers, where four of them encode information given by proprioception and the remaining three encode the spatial direction of the end-effector. The input layers are connected all-to-all with plastic synapses to four output layers which represent the motor commands to the joints. A bell-shaped distribution models the mean firing rate in the encoding scheme.

[1] In this paper, we use the term "layer" when referring to a group of input/output neurons.

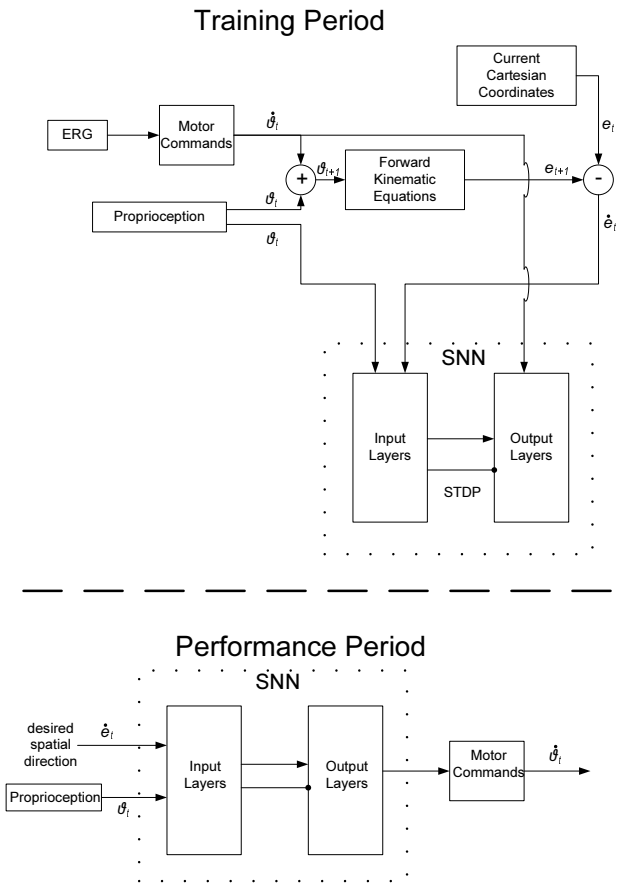

Fig. 3. A diagram of the system during the training and the performance period.

period, the network has to learn that when the arm lies on the joint configuration $\boldsymbol{\vartheta}_1$, the motor command vectors $\dot{\boldsymbol{\vartheta}}_1$ and $\dot{\boldsymbol{\vartheta}}_2$ move the end-effector in the spatial directions $\dot{\mathbf{e}}_1$ and $\dot{\mathbf{e}}_2$, respectively. This means that the synaptic weights should adapt so that, the *simultaneous* stimulation of $L^{input}_{i=1:4}(\boldsymbol{\vartheta}_1)$ and $L^{input}_{i=5:7}(\dot{\mathbf{e}}_1)$ will result in the activation of $L^{output}_{i=1:4}(\dot{\boldsymbol{\vartheta}}_1)$.[2] Similarly, the simultaneous stimulation of $L^{input}_{i=1:4}(\boldsymbol{\vartheta}_1)$ and $L^{input}_{i=5:7}(\dot{\mathbf{e}}_2)$ should result in the activation of $L^{output}_{i=1:4}(\dot{\boldsymbol{\vartheta}}_2)$. While this can be learned through STDP, a conflict is encountered if the network is subsequently called to learn that, when the arm rests in the new joint configuration $\boldsymbol{\vartheta}_2$, the motor command vector $\dot{\boldsymbol{\vartheta}}_1$ moves the end-effector in the spatial direction $\dot{\mathbf{e}}_2$. That is, the simultaneous stimulation of $L^{input}_{i=1:4}(\boldsymbol{\vartheta}_2)$ and $L^{input}_{i=5:7}(\dot{\mathbf{e}}_2)$ should activate the neurons in $L^{output}_{i=1:4}(\dot{\boldsymbol{\vartheta}}_1)$. If the synaptic weights in the network are modified to incorporate the last pattern, it is clear that the stimulation of $L^{input}_{i=1:4}(\boldsymbol{\vartheta}_1)$ and $L^{input}_{i=5:7}(\dot{\mathbf{e}}_2)$ would erroneously result in the activation of both neuron sets $L^{output}_{i=1:4}(\dot{\boldsymbol{\vartheta}}_1)$ and $L^{output}_{i=1:4}(\dot{\boldsymbol{\vartheta}}_2)$, while it should only activate $L^{output}_{i=1:4}(\dot{\boldsymbol{\vartheta}}_2)$, as given by the second training pattern. The output firing pattern $L^{output}_{i=1:4}(\dot{\boldsymbol{\vartheta}}_1)$ is erroneously activated because the sets of firing neurons $L^{input}_{i=1:4}(\boldsymbol{\vartheta}_1)$ and $L^{input}_{i=5:7}(\dot{\mathbf{e}}_2)$ are *individually* shown to be good predictors for $L^{output}_{i=1:4}(\dot{\boldsymbol{\vartheta}}_1)$, according to the first and third pattern under learning. To address this issue, we modify the population vector scheme and use many "bins" of neurons to represent a single value of a variable, which means many possible "central neurons". In this way, even when firing patterns have the above characteristic, the erroneous firing in the output layers can be avoided when at least one of the four central neurons representing $L^{input}_{i=1:4}(\boldsymbol{\vartheta}_1)$ is different in the first and second pattern, or $L^{input}_{i=5:7}(\dot{\mathbf{e}}_2)$ in the second and third pattern.

The aforementioned issues would not have been encountered if were following an alternative approach to representing the input patterns. As has been discussed, we use N independent neuronal layers to represent N variables, with the population of neurons in each layer encoding the value of a single variable. An alternative representation scheme that would not cause the issues discussed above would be to use a single N-dimensional array of neurons, where each instance of input pattern (*i.e.*, N-tuple) would be represented by stimulating a *unique* set of neurons. This representation however has the important drawback of poor scalability, since the population of neurons required increases exponentially with the number of variables represented. In particular, even if we had just 10 neurons representing a single variable, then this scheme would necessitate the use of 100000 neurons for 5 variables, and ten times this number if we were adding just a single variable. It is thus evident that such a representation can only be considered when the number of variables is small, and is not suitable for our task.

A. Neuron Model

Many models have been proposed in the literature in an attempt to simulate the behaviour of real neurons. An influential model was proposed by Hodgkin and Huxley [9] who translated their experimental observations on the giant axon of the squid into a set of nonlinear ordinary differential equations. While their model is considered to be biophysically accurate, their simulation is computationally expensive. An alternative model is based on integrate-and-fire neurons which carry much less computational burden. The shortcoming of this model however is its inability to reproduce the rich dynamics exhibited by cortical neurons. In this work, we simulate the individual neurons according to Izhikevich's "simple model" [10]. This model preserves the biologically realistic behaviour exhibited by the Hodgkin-Huxley model, and at the same time is computationally efficient as the integrate-and-fire model. The low computational cost is especially important when it comes to simulate large networks. The efficiency of the model relies on the fact that it uses only two equations and has only one non-linear term. In particular, the equations describing the model are given by:

$$\dot{v} = 0.04v^2 + 5v + 140 - u + I \quad (7)$$

$$\dot{u} = a(bv - u) \quad (8)$$

[2] $L_i(\theta_0)$ denotes the set of neurons in the i-th layer which represent the value θ_0.

with after-spike resetting:

$$\text{if } v \geq 30 \text{ then } \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{9}$$

where v denotes the neuron's membrane potential, I is the input current, and u is the variable that determines the recovery period of the neuron after spiking. The membrane recovery variable u provides negative feedback to the membrane potential v, and emulates the activation of K^+ ionic currents and the inactivation of Na^+ ionic currents. There are four parameters that alter the behaviour of a neuron, namely a, b, c and d, that can be set to accurately simulate a large variety of types of neurons. In particular, a determines the time scale of the recovery period, b represents the coupling between u and v, while c and d denote the after-spike reset values for v and u, respectively. Characteristic values of these parameters are $a = 0.02, b = 0.2, c = -65mV$ and $d = 2$. According to this model, a spike is produced when the membrane potential of a neuron reaches the threshold of 30 mV.

The simulation is run in discrete time, setting the time step to 1 msec. At each simulation step t, the incoming current $I(t)$ for a neuron i is updated based on the activity of a set of presynaptic neurons Q which are connected to neuron i with a conductance delay δ and fired at the time step $t - \delta$. In particular,

$$I(t) = I_b + \sum_{j \in Q} S_{i,j} F \tag{10}$$

where I_b is the base current, $S_{i,j}$ is the synaptic weight from neuron j to neuron i, and F is a scaling factor. In our simulation, $F = 0.2$ and $\delta = 1$ msec.

B. Learning Mechanism: Spike Timing-Dependent Plasticity

Spike Timing-Dependent Plasticity is regarded as a biologically plausible learning mechanism that modifies the synaptic strength between real neurons. Its exact form varies between different types of synapses and many models have been proposed ([11], [14]–[16]). Common differences between the various versions of STDP is the amount of change in weights, the dependence or not of the weight update on the current synaptic weight, and the time windows that are examined before and after the spike of the postsynaptic neuron. In this work, we use the symmetric version of STDP [17], which has been found in [6] to be robust to the temporal structure of the input patterns. In this version of STDP, the decision of whether a synapse should be potentiated or depressed does not depend on the temporal order of the events (arrival of the presynaptic spike at postsynaptic neuron before/after the firing of the postsynaptic neuron), but instead on their absolute time difference $\mid t_{post} - t_{pre} \mid$. That is, synapses are potentiated when they deliver spikes slightly before or after the firing of the postsynaptic neuron, while they get depressed when the time lag is greater (see Figure 4). This rule is described by:

$$\Delta S_{i,j} = A_{sym} \left(1 - \left(\frac{\Delta t}{\tau_a}\right)^2\right) e^{-\frac{|\Delta t|}{\tau_b}} \tag{11}$$

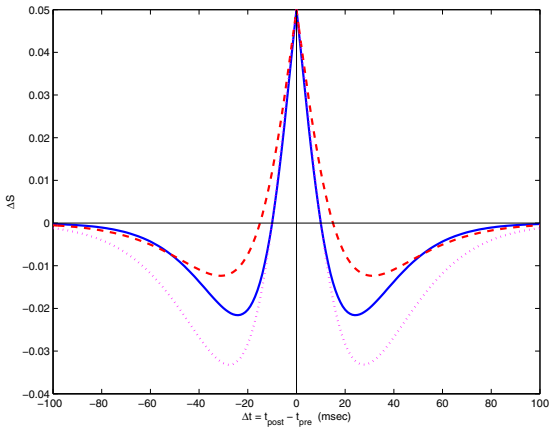

Fig. 4. Symmetrical STDP. The dotted line illustrates the symmetrical STDP rule for $\tau_a = 10$ msec and $\tau_b = 12$ msec, the dashed line for $\tau_a = 15$ msec and $\tau_b = 12$ msec, and the solid line for $\tau_a = 10$ msec and $\tau_b = 10$ msec. As can be observed, the solid line and the dotted line have the same positive time window (*i.e.*, the same τ_a), while the dashed line has the smallest depression area.

where $\Delta S_{i,j}$ denotes the weight update between the presynaptic neuron j and postsynaptic neuron i, A_{sym} is a coefficient that controls the magnitude of the synaptic change, τ_a determines the time window in which the incremental change of weights is positive, the ratio $\frac{\tau_a}{\tau_b}$ controls the balance between potentiation and depression, and $\Delta t = t_{post} - t_{pre}$. In the present work, we set $A_{sym} = 0.05, \tau_a = 10$msec, and $\tau_b = 10$msec. The time window between presynaptic and postsynaptic neurons that are included in STDP is set to 50 msec.

IV. Experiments

The proposed spiking neural network has been tested on controlling the kinematic model of the arm of a humanoid robot, called *iCub*. This robot has been developed as part of the RobotCub project[3] and has been designed to resemble the size of a 3.5 year old child. It has 53 degrees of freedom, weights around 22 kg, and is approximately 1 m tall. Each arm has seven DoFs, with three of them located on the shoulder, one DoF located on the elbow and three DoFs on the wrist. In the present work, we control the four upper DoFs of the iCub's arm, but the extension of the proposed spiking neural network to control all of the seven DoFs is straightforward.

During the training stage, a set of configurations of the arm are selected as "home" positions. The Endogenous Random Generator sends random motor commands, in the range of $[-5^o, 5^o]$ at each joint, and their effect on the spatial position of the end-effector is computed based on forward kinematic equations, according to the Denavit-Hartenberg parameters of the iCub's arm [18]. Each iteration includes four movements of the arm, and for each movement, the neurosimulation encodes the joint positions, the spatial direction of movement and the motor commands with a firing stimulus of 20 msec

[3]http://www.robotcub.org

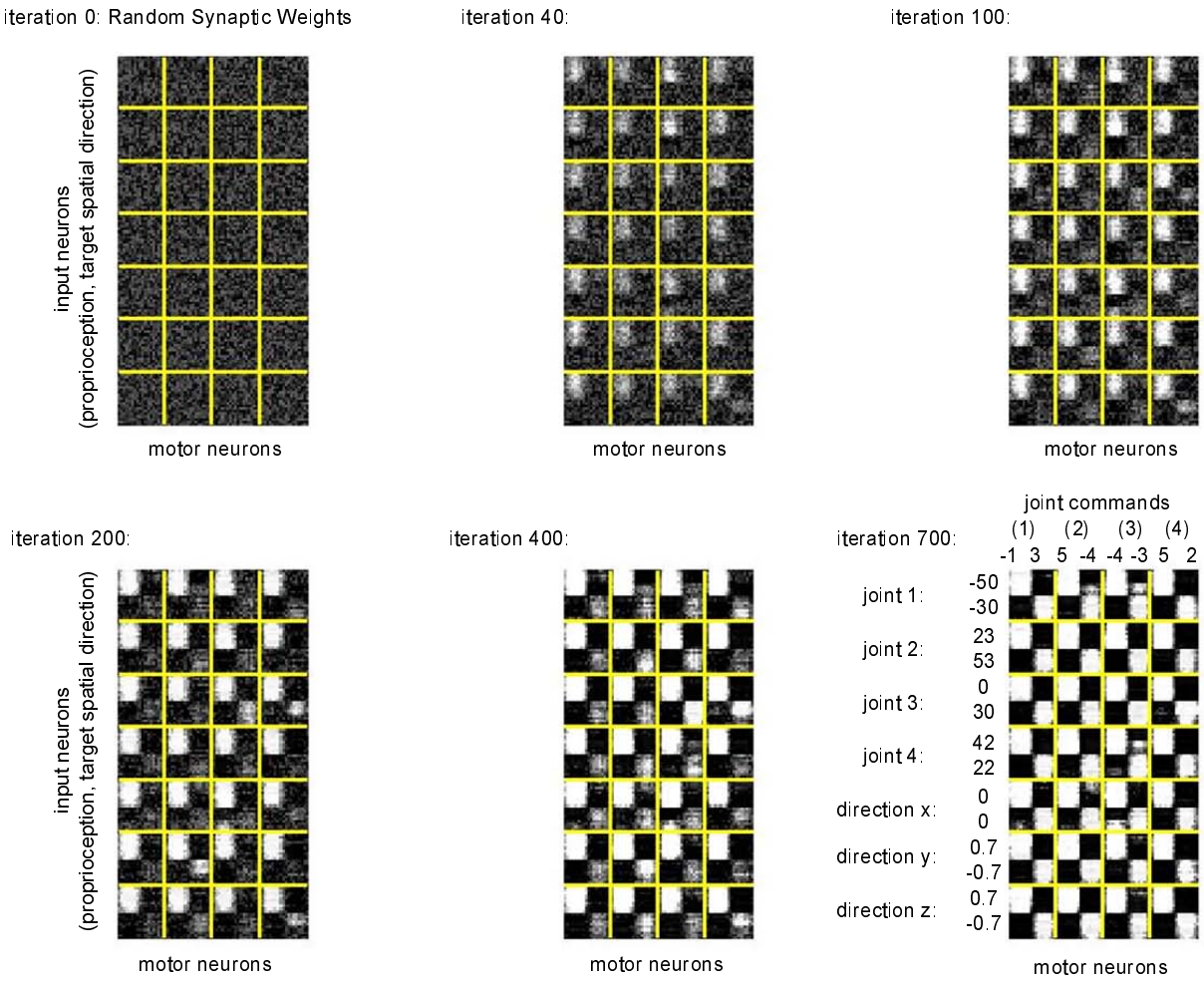

Fig. 5. The connectivity matrix between a set of input and motor neurons (see text). The matrix is shown for the iterations 0, 40, 100, 200, 400 and 700. Darker color indicates lower synaptic weights. As can be seen, the synaptic weights start from random values before training, and their values change to accommodate the correlation between input and output patterns. In the specific example, two different central neurons have been assigned to represent the value 0 in the fifth input layer that represents the spatial direction x, as has been explained in the text.

in the respective neuronal layers. An interval of 50 msec is set between the firing patterns that represent two consecutive movements. In the simulation, a background noise of 3 Hz is used in both input and output layers in order to weaken the synaptic weights between uncorrelated neurons.

The connectivity matrix for a subset of the synaptic weights between input and motor neurons during the training stage is shown in Figure 5, where darker colors represent weaker synapses. The set of input neurons included in the connectivity matrix shown represent the arm configurations $\boldsymbol{\vartheta}_1 = [-50^o, 23^o, 0^o, 42^o]$ and $\boldsymbol{\vartheta}_2 = [-30^o, 53^o, 30^o, 22^o]$, as well as the spatial directions $\dot{\mathbf{e}}_1 = [0, 0.7, 0.7]$ and $\dot{\mathbf{e}}_2 = [0, -0.7, -0.7]$. Similarly, the set of motor neurons included represent the motor commands $\dot{\boldsymbol{\vartheta}}_1 = [-1^o, 5^o, -4^o, 5^o]$ and $\dot{\boldsymbol{\vartheta}}_2 = [3^o, -4^o, -3^o, 2^o]$. We have selected this set of input and motor neurons for illustration, as the motor commands $\dot{\boldsymbol{\vartheta}}_1$ move the arm from the configuration $\boldsymbol{\vartheta}_1$ in the direction $\dot{\mathbf{e}}_1$, and the motor commands $\dot{\boldsymbol{\vartheta}}_2$ move the arm from the pose $\boldsymbol{\vartheta}_2$ in the direction $\dot{\mathbf{e}}_2$. Recall that we have 7 input layers and 4 output layers, and these layers are separated in the connectivity matrix with thick yellow lines. As can be seen in the upper-left corner of Figure 5, the synaptic weights are initially set to random values. As the training takes place, we can observe that progressively the synaptic weights between the correlated neurons $(\boldsymbol{\vartheta}_1, \dot{\mathbf{e}}_1)$ and $\dot{\boldsymbol{\vartheta}}_1$ increase, while the synaptic weights between the uncorrelated $(\boldsymbol{\vartheta}_1, \dot{\mathbf{e}}_1)$ and $\dot{\boldsymbol{\vartheta}}_2$ decrease. At the 700^{th} iteration, we can see that the correlation between $(\boldsymbol{\vartheta}_2, \dot{\mathbf{e}}_2)$ and $\dot{\boldsymbol{\vartheta}}_2$ has also been established.

After training, the network is able to generate those motor commands which, for a given arm configuration, move the end-effector in the desired spatial direction. In particular,

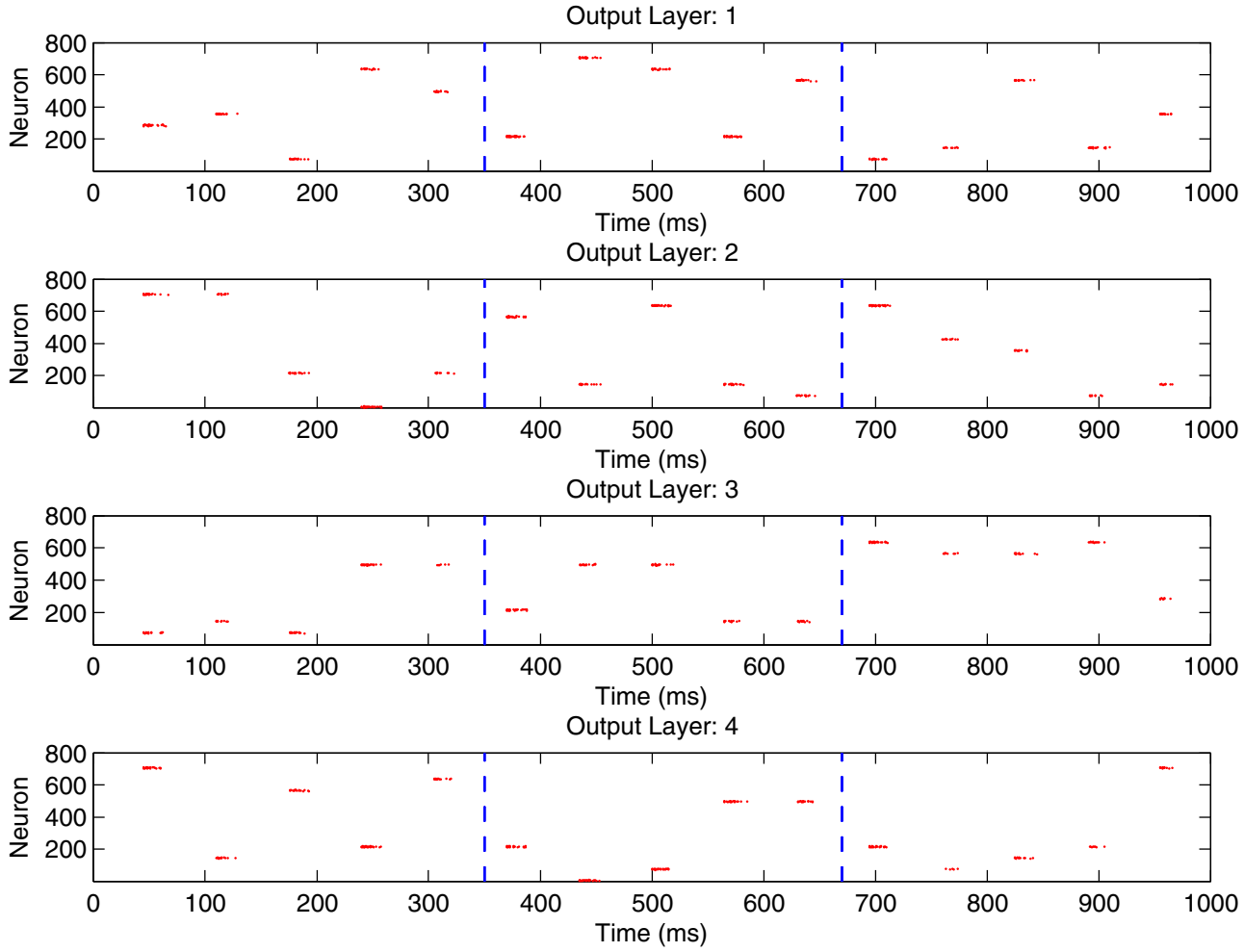

Fig. 6. Raster plot of the output layers. We consider three arm configurations, and for each configuration, we set five desired spatial directions. The time windows [0,350], [350, 670], and [670, 1000] refer to the three arm poses. Note that the first desired direction in each pose is the same, but different motor commands are required. The arm configurations considered are: $[-50^o, 23^o, 0^o, 42^o]$, $[-30^o, 53^o, 30^o, 22^o]$, $[-20^o, 65^o, 40^o, 60^o]$, while the spatial directions given are: [0 0.7 0.7; 0 1 0; 0 0 1; 0 -0.7 -0.7; 0 -1 0] (for the first arm pose), [0 0.7 0.7; 0.7 0 -0.7; 0.7 0.7 0; -0.7 -0.7 0; 0 -0.7 -0.7] (for the second arm pose), and [0 0.7 0.7; 0 1 0; 0.5 0.5 -0.7; 0 -0.7 -0.7; 0 -1 0] (for the third arm pose).

we consider three arm poses, which have also been used as "home" positions during the training stage, and set, for each pose, five desired spatial directions. The resulting raster plot of the output layers is shown in Figure 6, which is decoded into motor commands in order to move the end-effector in the desired spatial directions. It is worth noting that the same spatial direction in the three arm poses requires different motor commands, and thus necessitates the activation of different output neurons. This is achieved in the proposed network, even though there is a partial overlapping in the input stimuli due to the fact that the input layers $L^{input}_{i=5:7}$ encode the same spatial direction. The desired directions and the actual directions of movement produced by the decoded motor commands can be seen in Figure 7, where the mean difference is 7^o. The error can be decreased by using smaller discretization bins for the spatial direction during the training stage (currently, the bin size is 45^o). Finally, Figure 8 illustrates the joint angles of the arm that the spiking neural network provides when aiming to move the end-effector in a certain spatial direction (*i.e.*, [0 0.7 0.7]) for consecutive steps.

V. Conclusion

In this paper, we have presented a spiking neural network that autonomously learns to control a four degree-of-freedom robotic arm in three dimensional space. The neural network consists of approximately 12000 Izhikevich's neurons, and has a feed-forward architecture. The input layers of the network encode the joint positions of the arm and the desired spatial direction of the end-effector, and the output layers represent the corresponding motor commands. We have chosen this architecture due to the fact that the set of motor commands that drive the end-effector in a certain spatial direction is only valid in a local region of the joint space, and thus, the input firing pattern should encode the desired spatial direction in conjunction with the current joint configuration of the arm. The training takes place during a motor babbling

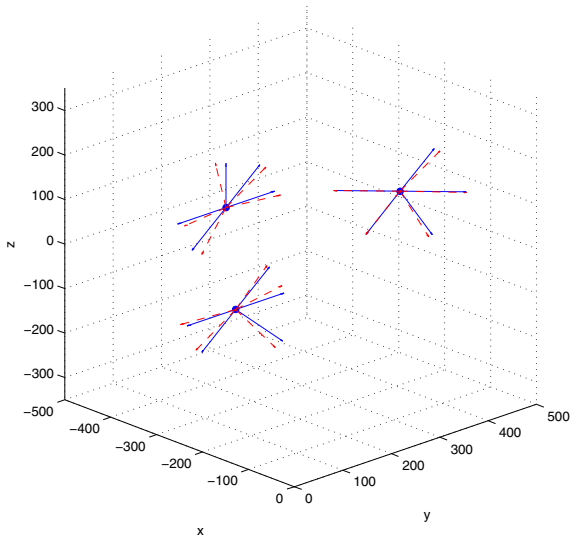

Fig. 7. The desired directions and the actual directions of movement produced by the decoded motor commands.

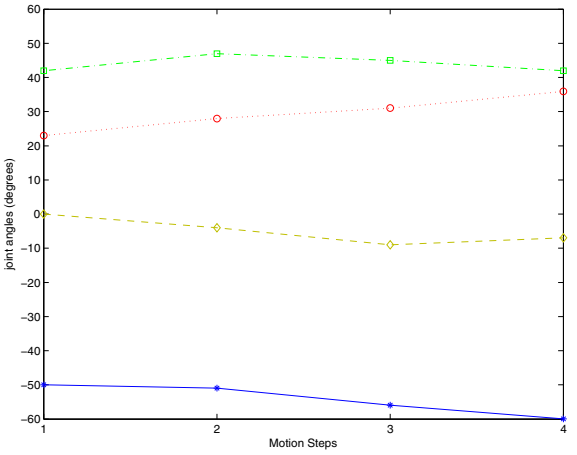

Fig. 8. The arm initially rests on the joint configuration $[-50^o, 23^o, 0^o, 42^o]$. The figure depicts the joint angles as they have been updated by the neural network when moving the end-effector towards the spatial direction [0 0.7 0.7]. Symbols '*', 'o', '◇' and '□' correspond to the first, second, third and fourth joint respectively.

period, and Spike Timing-Dependent Plasticity has been used as the learning mechanism. We have shown that this mechanism is able to temporally associate the input and output patterns, modify the synaptic weights accordingly, and train the network to perform the mapping from spatial commands to joint commands. An important feature of the proposed network is its scalability with respect to the number of degrees of freedom, as the population of neurons required increases linearly with the number of joints. The current implementation of the network is computationally costly (several seconds of processing time are required per arm movement). In future work, we aim to implement the proposed network using our GPU architecture [19] in order to achieve real-time performance.

ACKNOWLEDGMENT

This work was supported by an EPSRC Grant under Project Code EP/F033516/1.

REFERENCES

[1] D. Bullock, S. Grossberg and P.H. Guenther, "A Self-Organizing Neural Model of Motor Equivalent Reaching and Tool Use by a Multijoint Arm", *Journal of Cognitive Neuroscience*, vol. 5, no. 4, pp. 408–435, 1993

[2] P. Gaudiano and S. Grossberg, "Vector associative maps: Unsupervised real-time error-based learning and control of movement trajectories", *Neural Networks*, vol. 4, no. 2, pp. 147–183, 1991

[3] S. Grossberg, "Some nonlinear networks capable of learning a spatial pattern of arbitrary complexity", *Proceedings of the National Academy of Sciences of the United States of America*, vol. 59, no. 2, pp. 368–372, 1968

[4] G. Asuni, G. Teti, C. Laschi, E. Guglielmelli and P. Dario, "A Robotic Head Neuro-controller Based on Biologically-Inspired Neural Models", *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2362–2367, 2005

[5] Q. Wu, T.M. McGinnity, L. Maguire, A. Belatreche and B. Glackin, "2D co-ordinate transformation based on a spike timing-dependent plasticity learning mechanism", *Neural Networks*, vol. 21, no. 9, pp. 1318–1327, 2008

[6] A.P. Davison and Y. Frégnac, "Learning Cross-Modal Spatial Transformations through Spike Timing-Dependent Plasticity", *Journal of Neuroscience*, vol. 26, no. 21, pp. 5604-5615, 2006

[7] R. Borisyuk, Y. Kazanovich, D. Chika, V. Tikhanoff and A. Cangelosi, "A neural model of selective attention and object segmentation in the visual scene: An approach based on partial synchronization and star-like architecture of connections", *Neural Networks*, vol. 22, no. 5-6, pp. 707-719, 2009

[8] S.G. Wysoski, L. Benuskova and N. Kasabov, "Fast and adaptive network of spiking neurons for multi-view visual pattern recognition", *Neurocomputing*, vol. 71, pp. 2563–2575, 2008

[9] A.L. Hodgkin and A.F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve", *Journal of Physiology*, vol. 117, no. 4, pp. 500-544, 1952

[10] E.M. Izhikevich, "Simple model of spiking neurons", *IEEE Transactions on Neural Networks*, vol. 14, pp. 1569-1572, 2003

[11] S. Song and L.F. Abbott, "Cortical development and remapping through spike timing-dependent plasticity", *Neuron*, vol. 32, pp. 339-350, 2001

[12] A. Pouget and P.E. Latham, "Population codes", *The Handbook of Brain Theory and Neural Networks*, 2nd edition. M. Arbib, MIT Press, 2003

[13] A.P. Georgopoulos, A.B. Schwartz and R.E. Kettner, "Neuronal population coding of movement direction", *Science*, vol. 233, no. 4771, pp. 1416–1419, 1986

[14] S. Song, K.D. Miller and L.F. Abbott, "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity", *Nature Neuroscience*, vol. 3, pp. 919–926, 2000

[15] G. Bi and M. Poo, "Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type", *Journal of Neuroscience*, vol. 18, no. 24, pp. 10464–10472, 1998

[16] D.E. Feldman, "Timing-based LTP and LTD at vertical inputs to layer II/III pyramidal cells in rat barrel cortex", *Neuron*, vol. 27, pp. 45–56, 2000

[17] M.A. Woodin, K. Ganguly and M. Poo, "Coincident Pre- and Postsynaptic Activity Modifies GABAergic Synapses by Postsynaptic Changes in Cl^- Transporter Activity", *Neuron*, vol. 39, pp. 807-820, 2003

[18] B. Siciliano and O. Khatib, "Springer Handbook of Robotics", Springer-Verlag, 2008

[19] A. Fidjeland, E.B. Roesch, M.P. Shanahan and W. Luk, "NeMo: A platform for neural modelling of spiking neurons using GPUs", *20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, 2009