Bioinspiration & Biomimetics

PAPER

Differential mapping spiking neural network for sensor-based robot control

RECEIVED
23 December 2020

ACCEPTED FOR PUBLICATION
11 March 2021

PUBLISHED
2 April 2021

Omar Zahra[1], Silvia Tolu[2] and David Navarro-Alarcon[1,*]

[1] The Hong Kong Polytechnic University, Hong Kong Special Administrative Region of China
[2] Technical University of Denmark, Denmark
[*] Author to whom any correspondence should be addressed.

E-mail: david.navarro-alarcon@polyu.edu.hk

Keywords: robotics, visual servoing, sensor-based control, spiking neural networks

Supplementary material for this article is available online

Abstract

In this work, a spiking neural network (SNN) is proposed for approximating differential sensorimotor maps of robotic systems. The computed model is used as a local Jacobian-like projection that relates changes in sensor space to changes in motor space. The SNN consists of an input (sensory) layer and an output (motor) layer connected through plastic synapses, with inter-inhibitory connections at the output layer. Spiking neurons are modeled as Izhikevich neurons with a synaptic learning rule based on spike timing-dependent plasticity. Feedback data from proprioceptive and exteroceptive sensors are encoded and fed into the input layer through a motor babbling process. A guideline for tuning the network parameters is proposed and applied along with the particle swarm optimization technique. Our proposed control architecture takes advantage of biologically plausible tools of an SNN to achieve the target reaching task while minimizing deviations from the desired path, and consequently minimizing the execution time. Thanks to the chosen architecture and optimization of the parameters, the number of neurons and the amount of data required for training are considerably low. The SNN is capable of handling noisy sensor readings to guide the robot movements in real-time. Experimental results are presented to validate the control methodology with a vision-guided robot.

1. Introduction

Sensor-guided object manipulation is one of the most fundamental tasks in robotics, with many possible approaches to perform it [1]. Conventional methods typically rely on mathematical modeling of the observed end-effector pose and its related joint configuration. These methods provide accurate solutions, however, they require exact knowledge of the analytical sensor-motor relations (which might not be known); furthermore, these conventional methods are generally not provided with adaptation mechanisms to cope with uncertainties/changes in the sensor setup.

Among the many interesting cognitive abilities of humans (and animals, in general) is the motor babbling process that leads to the formation of sensorimotor maps [2]. By relying on such adaptive maps, humans can learn to perform many motion tasks in an efficient way. These advanced capabilities have motivated many research studies that attempt to artificially replicate such skills in robots and machines [3, 4]. Our aim in this work is to develop a bio-inspired adaptive computational method to guide the motion of robots with real-time sensory information and a limited amount of data.

Data-driven computational maps have been previously built for approximating unknown sensorimotor relations [5–7]. One common limitation of these classical approaches is the demand for a high number of training data points and high computational power, which is impractical in many cases.

By drawing inspiration from the central nervous system, artificial neural networks (ANN) have been built and used for many decades. It started with the McCulloch–Pitts model as the first generation of ANN by using binary computing units [8], followed by the second generation utilizing mainly the sigmoidal (or tanh) activation function to make it more capable of approximating non-linear functions [9].

Taking one step further, spiking neuronal network (SNN) (representing the third generation of ANNs) are designed to incorporate most of the neuronal dynamics which provide them with more complex and realistic firing patterns based on spikes [10]. These spikes are generated based on functions relating the current and membrane potential in the neuronal units, unlike the activation functions used to generate the output in non-spiking neurons. In this work, we have built an adaptive SNN-based model to guide robots with sensory feedback.

The main exclusive property of SNN is the incorporation of a temporal dimension; note that the relative timing of spikes (and spike sequences) enables the encoding of useful information. The temporal dimension allows a single spiking neuron to achieve a task carried out by hundreds of non-spiking neurons in some cases [10]. This is supported by observations in simple living organisms, as insects and worms, which can perform complex tasks by relying only on few neurons [11, 12]. As in real biological systems, SNNs hold an advantage for real-time processing (as concluded in [13] for the visual system) and multiplexing of information (such as amplitude and frequency in the auditory system [14]). For robotic applications, the SNN allows building computational models of brain regions to imitate intelligent behavior in living organisms to a great extent, and in some cases even reveal the mysteries of the inner workings of the brain [15–17]. The currently rising neuromorphic chips allow real-time operation of such models while conserving power greatly compared to conventional systems [18]. While an SNN allows building a more biologically plausible system, its complexity makes it more difficult to predict and analyze the behavior of the system. To this end, various methods can be used, e.g. simplifying the network's mathematical description [19] or applying techniques for tuning the parameters to achieve the desired performance [20]. The latter approach is the one addressed in this work.

Several studies have provided examples of the application of SNN in robotics [21]. However, in most studies, robots were only controlled in a simulation environment [22]. The main reason is due to the large size of the neural networks used in these works, which makes it impractical for real-time operations. In [20], a cognitive architecture is used for controlling a robotic hand to perform grasping motions. In [23], an SNN was used for learning motion primitives of a robot to move in three axes (left–right, up–down, and far–near). In [24], a self-organizing architecture was used to build an SNN representing spatio-motor transformations (i.e., kinematics) of a two degrees of freedom (DOF) robot. Despite its good performance and biological plausibility, the network's size and limited scalability make it impractical for real-time control.

In [25], an SNN with learning based on spike timing-dependent plasticity (STDP) was used to build the kinematics of a robot. Although the synaptic connections illustrate the ability to approximate the kinematic relation, the approximation error is not evident. Additionally, an intermediate layer is required to scale up the dimensions of the input sensory data, and consequently, the relative computational power.

In this paper, we propose an SNN-based control architecture to guide robots with sensor feedback. Our proposed neural controller adaptively builds a differential map that correlates end-effector velocities (as measured by an external vision sensor) with joint angular velocities. In other words, it effectively works as a local Jacobian-like transformation between different (sensor-motor) spaces. The proposed network has the following valuable features:

- Inter-inhibitory connections at the output layer exhibiting a winner takes all effect.
- Biologically plausible tools as building blocks.
- A proposed guideline for network parameters tuning.

To the best of the authors' knowledge, this is the first study to report an SNN-based method capable of forming the sensor-motor differential map in a computationally efficient way [resulting from an intuitive guideline to adjust its parameters along with the particle swarm optimization (PSO)]. This leads to a great reduction in the number of neurons compared to previous works in the literature [20, 23, 26] by more than 10 folds, and thus a real-time operation even with moderate computational power available. Therefore, this study contributes to:

(a) Exploiting the capabilities of a two-layer SNN with a minimalistic number of neurons.
(b) Validating the proposed claims through a detailed experimental study with a robotic platform performing vision-guided manipulation tasks.

The rest of this paper is structured as follows: section 2 describes the developed spiking neural network; section 3 presents the verification of the method proposed through a dummy test; section 4 presents the experimental results; section 5 discusses the methods and results; section 6 gives the conclusions.

2. Methods

Many studies have suggested that humans use internal models to represent perception and action [27–29]. Some researchers have built computational models of brain areas (e.g., the cerebellum) responsible for the generation and coordination of fine motor actions [30, 31].

Unlike in traditional robot control (where sensory transformations to motion commands are solved analytically), in the human brain (motor cortex) the sensorimotor relations are encoded by specific neural

circuits. To carry out a typical (eye-to-hand) visual servoing task, we must first establish the kinematic relation between the joint velocities and the measured end-effector motion [32]. This model can be formulated for m DOFs and n Cartesian DOFs as $\dot{x} = J(\theta)\dot{\theta}$, such that:

$$J(\theta) = \begin{bmatrix} \frac{\partial x_1}{\partial \theta_1} & \cdots & \frac{\partial x_1}{\partial \theta_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial x_n}{\partial \theta_1} & \cdots & \frac{\partial x_n}{\partial \theta_m} \end{bmatrix}, \quad (1)$$

where $\dot{x} \in \mathbb{R}^n$, $\theta \in \mathbb{R}^m$, $\dot{\theta} \in \mathbb{R}^m$ and $J(\theta) \in \mathbb{R}^{n\times m}$ are the (observed) spatial velocity, joint angles, joint velocities, and the Jacobian matrix, respectively (without loss of generality, we assume that $m \geqslant n$).

For velocity-based control, it is necessary to estimate the joint velocities to achieve a certain spatial velocity. This expression is denoted as:

$$\dot{\theta} = J^{\#}(\theta)\dot{x}, \quad (2)$$

where $J^{\#}$ is the pseudo-inverse of the J. In this work, a differential mapping spiking neural network (DMSNN) is proposed to build a computational model analogous to the Jacobian transformation relating changes in 1 joint-space DoF to 1 task-space DoF.

As shown in figure 1, the network is first subjected to the training phase in which both sensory readings and motor commands are fed to the network through motor babbling by executing random motions. After training the network for several iterations, the differential mapping is formed by the modulation of the connection between the input and output layers. Then, the network can be used during the control phase to guide the robot through the estimation of the required motor command to reach the desired target by feeding the sensory information to the input layer. This can be seen as an approximation of the motor cortex which is responsible for converting the desired motion into a motor command to be executed by other regions in the brain [33]. Details of the proposed network are described in the rest of this section.

2.1. Network layout

Network layout of the proposed neural controller is shown in figure 2, where each dimension of sensory input (joint angles and spatial velocity) and motor output (joint velocity) is represented by a one-dimensional array (assembly) of neurons. The complete network consists of $n + 2m$ assemblies of neurons. The input sensory layer consists of assemblies $l^{\theta_{1:m}}$ encoding the joint angles and $l^{\dot{x}_{1:n}}$ encoding the spatial velocity, while the output motor layer consists of $l^{\dot{\theta}_{1:m}}$ encoding the joint velocity controls. Each sensory neuron is connected through an excitatory and inhibitory synapse (which is omitted in figure 2 for clarity) to each motor neuron. This acts as a substitute for adding an inhibitory interneuron and allows for stable learning dynamics while avoiding an unbounded increase in connection strength and neuron activity. Additionally, each of the neurons encoding the output at each assembly (dimension) $\dot{\theta}_{1:m}$ is connected to the neurons of the same assembly through nonplastic inhibitory connections:

$$\tilde{\varepsilon}_{kj} = \exp\left(\frac{-(k-j)^2}{(\sigma_n N_l)^2}\right) - 1, \quad (3)$$

where $\tilde{\varepsilon}_{kj}$ denotes the strength of the *non-plastic* connection between neurons k and j, σ_n is the standard deviation, and N_l is the number of neurons in the assembly. Therefore, the further the neuron is, the stronger the inhibition activity is. This approximates the behavior of a winner-take-all effect, but with a change in the inhibitory value depending on the proximity to the winner neuron (ensuring a continuous and more robust output).

2.2. Training phase and control phase

For the proposed network to form the desired differential map, the information needs to be input/encoded into the network and extracted/decoded in a proper way. To be able to convert the signals from and to the network properly, the encoders (converting signals to spikes) and decoders (converting spikes to signals) are used. The input to the sensory layer (during the training and control phases) and motor layers (during the training phase only) is calculated for each neuron based on its preferred (central) value ψ_c^i as shown in figure 3. Thus, the tuning curve for the encoders is chosen to be the Gaussian distribution. The input current to a neuron i for a certain input can be formulated as:

$$\gamma_i = A \exp\left(\frac{-\|\psi - \psi_c^i\|^2}{2\sigma^2}\right) \quad (4)$$

where ψ is the input value, A is the amplitude of the input current, and σ is calculated based on the number of neurons per layer N_l, and the range of change of the variable to be encoded from Ψ_{min} to Ψ_{max}. This leads to the contribution of the whole layer to encode a particular value (a process that can be interpreted as 'population coding' [34]). The value of A is chosen based on the neuron parameters and different values A_s and A_m are assigned for the sensory and motor layers, respectively. The choice of A_s and A_m along with the neuron parameters allows to have a controlled firing activity and hence a controlled learning process.

To get the estimated output from the network, a proper decoding function has to be chosen. Among the various decoding methods, the central neuron voting scheme is selected to calculate the decoded value corresponding to the firing rate of all the neurons of a layer. This can be modeled for a specific time

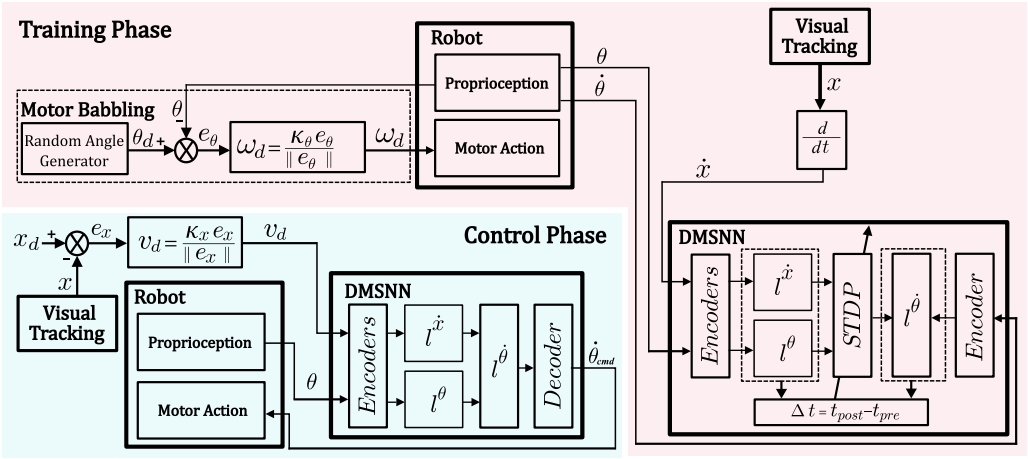


Figure 1. A schematic diagram for both the training and control phases for the DMSNN. The signals introduced to each neuron assembly are depicted. During the training phase, the robot's motion is guided by motor babbling in joint space. During the control phase, the robot's motion is guided by motor commands decoded from the activity of motor neurons at the output layer.

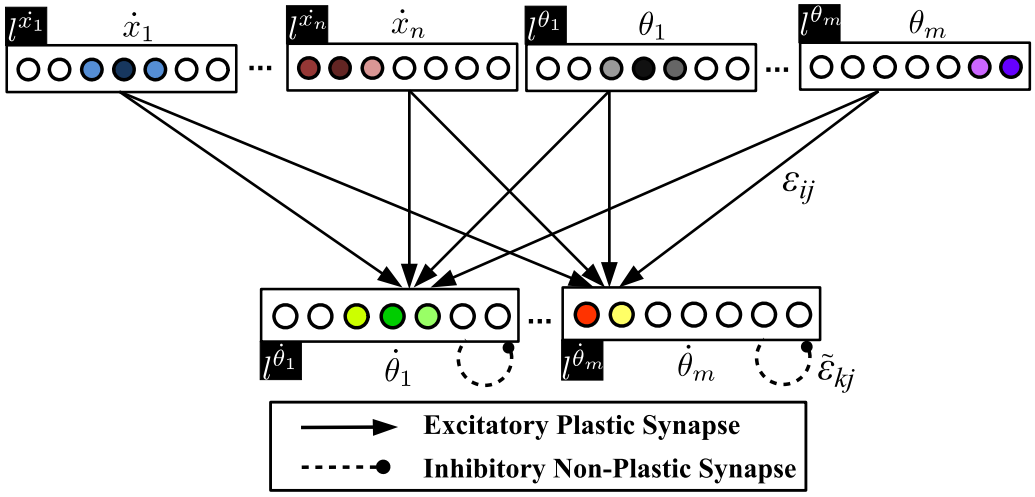


Figure 2. The layout of the proposed spiking neural network. Input (sensory) neurons are connected to output (motor) neurons through excitatory and inhibitory plastic synapses ε_{ij}; neurons in each motor assembly are interconnected through inhibitory non-plastic synapses $\tilde{\varepsilon}_{ij}$.

window, as follows:

$$\psi_{\text{est}} = \frac{\Sigma \psi_{\text{c}}^{i}.\alpha_i}{\Sigma \alpha_i}, \tag{5}$$

where ψ_{c}^{i} is the central value of neuron i firing with a rate α_i in the assembly l^{Ψ}, and ψ_{est} is the estimated (decoded) value of the output [34]. Hence, the decoder acts to estimate the value encoded by the activity of the whole assembly, which makes it act as a complement for the chosen encoding method.

Figure 1 depicts a schematic diagram of the proposed SNN-based method. Firstly, the motor babbling process initiates the training phase by providing the motor commands (joint velocity command ω_{d}) for the robot to move linearly in the joint space through numerous random targets, as it is formulated in:

$$\omega_{\text{d}} = \kappa_{\theta} \frac{e_{\theta}}{||e_{\theta}||}, \tag{6}$$

where $e_{\theta} = \theta_{\text{d}} - \theta$ is the error between the randomly generated desired joint angles (θ_{d}) and the current joint angles (θ).

This joint velocity command is scaled by the gain $\kappa_{\theta} > 0$, which is also varied randomly within a certain range during the babbling, to generate richer training data. During babbling motion, sensory information and motor commands are fed into the network to guide the modulation of the plastic synapses between the input and output layers through STDP. Sensory information is introduced from the proprioception (θ and $\dot{\theta}$) and the external sensor (the visual velocity $\dot{x}$), which are then encoded to be introduced to the corresponding assemblies. The difference in time of spikes generated in sensory and motor neurons $\Delta t = t_{\text{post}} - t_{\text{pre}}$ controls the amount of change in the synapses' strength. After a sufficient number of iterations (decided depending on the size of the

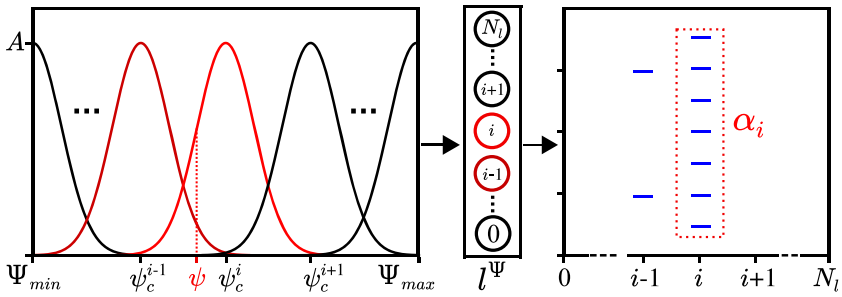


Figure 3. The encoder first converts the input value ψ by comparing it to the Gaussian curves of each neuron. Then, the output of the encoder γ is introduced to the layer l^Ψ to generate a set of spikes per iteration α by each neuron.

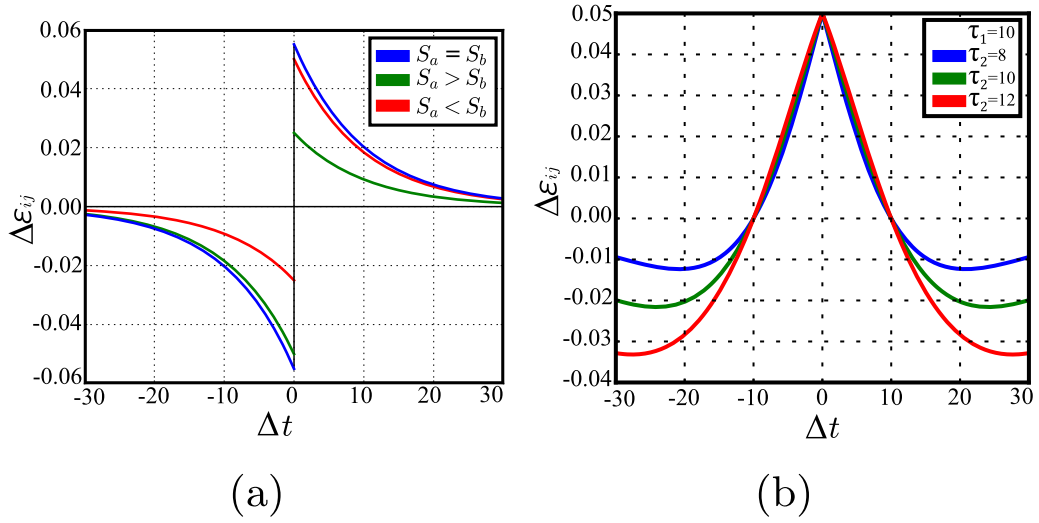


Figure 4. (a) Asymmetric and (b) symmetric learning STDP rules. For asymmetric learning, S_a and S_b are plotted at different values while keeping τ_a and τ_b constant. For the symmetric one, τ_2 is plotted at different values while keeping τ_1 constant.

workspace, the learning rate, and desired precision as explained in subsection 2.6) the robot is ready to perform a sensor-guided motion task.

$$v_{\mathrm{d}} = \kappa_x \frac{e_x}{\|e_x\|}, \tag{7}$$

where $e_x = x_d - x$ denotes the feedback spatial error and $\kappa_x > 0$ a variable gain that regulates the velocity by which the robot is driven towards the target x_d from the current position x. Finally, the motor command $\dot{\theta}_{\mathrm{cmd}}$ is decoded from the spikes generated at the output assemblies $l^{\dot{\theta}_{1:m}}$ as follows:

$$\dot{\theta}_{\mathrm{cmd}} = \psi_{\mathrm{est}}. \tag{8}$$

This value is then fed into the robot servo controller to guide its motion towards x_d as shown in figure 2.

2.3. Neuron model

Among the models available for spiking neurons is the Hodgkin and Huxley model that explains the mechanism of triggering an action potential and its propagation [35]. The mathematical model is composed of a set of nonlinear differential equations, which makes it computationally intense. However, this model is the most biologically plausible among all available models.

Another model that is widely used is the leaky integrate and fire model (LIF). In this model, the behavior of a neuron is approximated as a simple RC circuit with a low-pass filter and a switch with a thresholding effect [36]. The RC circuit is first charged by an input current, which makes the voltage at the capacitor to increase until it reaches the threshold value. The switch opens and lets a pulse (spike) to be generated; the capacitor then starts to build up again. This model is simple and has a low computational cost, however, compared to Hodgkin–Huxley's model, it is not very biologically plausible.

In 2003, Izhikevich developed a model that can reproduce the various firing patterns recorded by different neurons in different brain regions [37]. This model is chosen for our study as it holds a balance between a reasonable computational cost while preserving the biological plausibility. The model can be described by the following set of differential equations:

$$\dot{v} = f(v,u) = 0.04v^2 + 5v + 140 - u + I \tag{9}$$

$$\dot{u} = g(v,u) = a(bv - u). \tag{10}$$

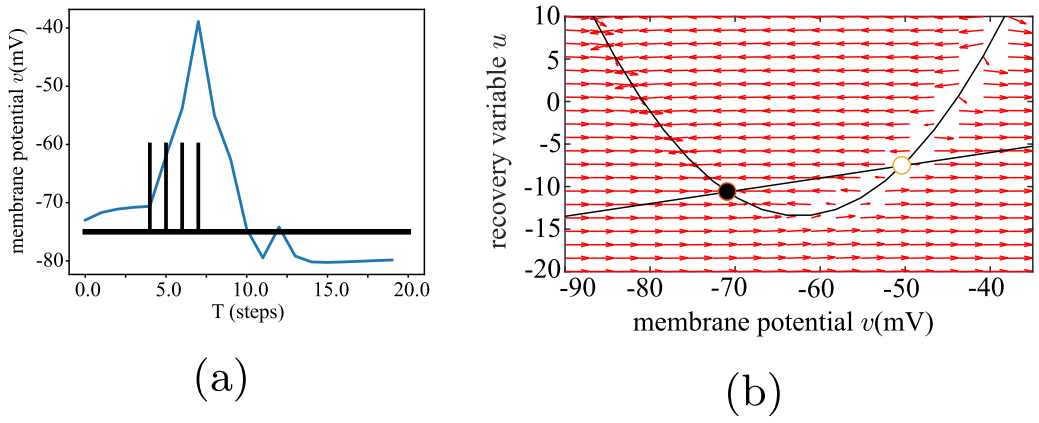


Figure 5. (a) Computational and (b) analytical analysis of a spiking neuron. In (a) an integrator neuron receives spikes from four neurons to cross a threshold value and trigger a spike, while in (b) a phase portrait of a spiking neuron is plotted.

After a spike occurs, the membrane potential is reset as:

$$\text{if } v \geqslant 30\,\text{mV}, \quad \text{then } v \leftarrow c,\ u \leftarrow (u+d), \quad (11)$$

where v is the membrane potential and u is the membrane recovery variable as shown in figure 5(b). The parameter a determines the time constant for recovery, b determines the sensitivity to fluctuations below the threshold value, c gives the value of the membrane potential after a spike is triggered, and d gives the value of the recovery variable after a spike is triggered. The term I represents the summation of the external currents introduced.

2.4. Synaptic connections

Synapses are the connections that transmit signals between two neurons. Let us denote by ε_{ij} the weight/strength of the synapse. The transmission acts only in one direction, such that signals are carried from the ith presynaptic to the jth postsynaptic neuron. The information transmitted through synapses is usually encoded in the form of spikes (or action potentials). Different theories have been presented for the way of encoding and decoding such information in our brain [34]. The synaptic connections can either be excitatory or inhibitory, and plastic or nonplastic. Excitatory synapses are the connections that are more likely to increase the activity of postsynaptic neurons with the increase in the activity of the presynaptic neuron, while inhibitory synapses decrease that likelihood. The synapses connecting the input layer to the output layer are plastic (which means that its strength is subject to change). Let us denote by $\Delta\varepsilon_{ij}[t]$ the synapse's change of strength at the time instance t, which satisfies the following discrete update rule:

$$\varepsilon_{ij}[t+1] = \varepsilon_{ij}[t] + \Delta\varepsilon_{ij}[t]. \quad (12)$$

One of the first learning rules to update the synaptic weights is the Hebbian learning rule [38], whose basic formulation is:

$$\Delta\varepsilon_{ij} = \eta a_i a_j \quad (13)$$

where the scalar η is the learning rate, a_i and a_j are the activities (or average firing rates) of the pre and postsynaptic neurons, respectively. This rule strengthens the connections between strongly correlated variables and has been shown to perform principal component analysis (PCA) [39]. However, this type of learning does not take into consideration the *time difference* between spikes, which is the main feature of SNNs.

Another learning rule that is more appealing from a biological perspective is STDP, where potentiation (increase) or depression (decrease) in the strength of the connections is dependent upon the relative timing of spikes that occur in presynaptic and postsynaptic neurons [40]. STDP is considered as a temporal form of Hebbian learning [41], e.g. in [42], it is shown that it can perform 'kernel spectral component analysis', which resembles PCA. This attribute makes it suitable for mapping two spaces while successfully updating the synaptic weights.

In the literature [43], two common patterns for STDP are either as symmetric [44] or antisymmetric [45], as depicted in figure 4. The antisymmetric model can be formulated as:

$$\Delta\varepsilon_{ij} = \begin{cases} -S_a \exp\left(-\Delta t/\tau_a\right) & \Delta t \leqslant 0 \\ S_b \exp\left(-\Delta t/\tau_b\right) & \Delta t > 0 \end{cases} \quad (14)$$

where S_a and S_b are coefficients that control the magnitude of the synaptic depression and potentiation, respectively, where τ_a and τ_b determine the time window through which depression and potentiation occur. The asymmetric STDP is thus suitable for a learning process whenever the sequence of signals matters, e.g., if a spike arrives from the presynaptic neuron before the spike from a postsynaptic neuron (which results in a positive value for Δt, and thus the synapse's weight is potentiated).

Algorithm 1. Network parameters tuning.

Input
- n, m = No. of dimensions of spaces to be encoded
- N_{l} = No. of neurons to encode each dimension
- Itr^* = No. of iterations to build the map

Output
- a, b, c, d = Neurons parameters
- I^* = Minimum input to have continuous spikes
- $C_{\mathrm{I}}, C_{\mathrm{E}}$ = Maximum strength of inhibitory and excitatory connections, respectively
- S = Learning rate for STDP based synaptic connections

Routine
1: Assign initial values for neuron parameters a, b, c, d
2: **While** $\xi \geqslant e_{\mathrm{th}}$ **do**
3: Build the neuron model $f(v, u)$ and $g(v, u)$
4: Solve for nullclines at $f(v, u) = 0$ and $g(v, u) = 0$
5: Get v^* and u^* where $f(v^*, u^*) = g(v^*, u^*) = 0$
6: Evaluate $L(v^*, u^*)$ to check stability
7: Obtain I^* and hence C_{E}
8: Set S for given Itr^*
9: Update values for a, b, c, d
10: Train the network
11: Perform several target reachings to obtain value of ξ
12: **end while**

The symmetric learning model is the most appropriate in this case given the continuous firing at both input and output layers. Furthermore, a symmetric STDP rule (with different reward modulated versions) was reported to be observed in the hippocampus and prefrontal cortex in several studies [46–48] and was studied in [49]. The symmetric STDP rule can be described by:

$$\Delta\varepsilon_{ij} = S\left(1 - \left(\frac{\Delta t}{\tau_1}\right)^2\right)\exp\left(\frac{|\Delta t|}{\tau_2}\right) \qquad (15)$$

where S is a coefficient that controls the magnitude of the synaptic change, the ratio between τ_1 and τ_2 decides the time window through which potentiation and depression occurs, and Δt is the difference between the timing of spikes at post t_{post} and pre t_{pre} synaptic neurons. In this study, we use $S = 0.05$, $\tau_1 = 20$ ms and $\tau_2 = 18$ ms. In this learning model, the change in synapse's weight is controlled by the absolute value of Δt, but not the sign, i.e. the sequence of firing. The chosen time window for the pre and postsynaptic neurons for STDP is 30 ms, which means as long as $-30 \leqslant \Delta t \leqslant 30$, it will still contribute to the modification of the synaptic strength.

With the absence of the hidden layer(s), the nonlinearity in the neuronal units, as well as the features of STDP, make it possible to learn the differential map. Such approach is supported by previous studies (see [20, 26]), as illustrated in section 4.

To perform the vision-guided motion task with the proposed network, a careful setting of its parameters is needed, as explained in the next section.

2.5. Manual tuning of the network parameters

Tuning the various parameters of the neurons and synapses is a difficult task. Some basic rules can be used to guide the trial and error approach to choose a set of appropriate values. For example, as u is the membrane recovery variable, it is responsible for the delivery of negative feedback to v, such that it resists change of the value of v. The network's parameters a, b, c and d tune how v and u change and interact together over time.

Figure 5(b) shows the phase portrait of the output neurons, as it plots the recovery potential versus the membrane potential. The two black curves represent the *nullclines* for both v and u, i.e. the line along which partial derivative equals zero which separates the planes of variation of v and u. To obtain these nullclines and analyze the system, equations (9) and (10) are set equal to zero to obtain lines along which there is no change in v and u, respectively. The intersection of these nullclines forms attractor (stable) points or repeller (unstable) points [50]. From the location of attractors and repellers, the stability regions can be concluded. The equilibrium points (v^*, u^*) are obtained by solving the equations of the parabola and line obtained from equating $f(v, u)$ and $g(v, u)$ together. The stability of these points is determined by the eigenvalues of the linearization matrix L:

$$L(v^*, u^*) = \begin{bmatrix} \frac{\partial f}{\partial v}(v^*, u^*) & \frac{\partial f}{\partial u}(v^*, u^*) \\ \frac{\partial g}{\partial v}(v^*, u^*) & \frac{\partial g}{\partial u}(v^*, u^*) \end{bmatrix} = \begin{bmatrix} 0.08v^* + 5 & -1 \\ ab & -a \end{bmatrix}. \qquad (16)$$

As the v-nullcline shifts upward, the attractor and repeller annihilate each other and merge into a saddle; any further upward shift leads to the disappearance of the saddle point.

As shown in algorithm 1, to use an SNN to model a certain system, initial values should be assigned for neuron parameters a, b, c, d based on

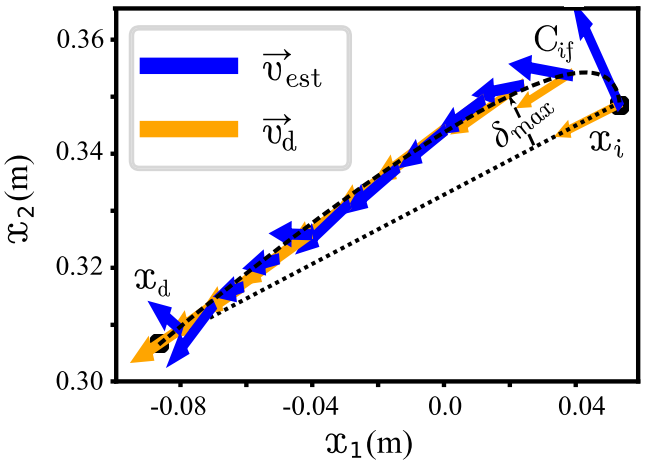


Figure 6. A planar robot driven by the DMSNN, initially at x_i reaching to a target at x_d and at each time step both the actual velocity ($\vec{v}_{est}$) and the desired velocity vector ($\vec{v}_d$) are calculated.

their role within the neuronal layer. In this work, the motor neurons are modeled as *integrator neurons* as it acts as a coincidence detector, such that it triggers a spike by accumulating closely timed signals as illustrated in figure 5(a). The initial values for the neuron parameters are $a = 0.02$, $b = -0.1$, $c = -55$, and $d = 6$. The sensory neurons are modeled as *fast spiking* neurons providing high frequency spikes and initialized with $a = 0.1$, $b = 0.2, c = -65$, and $d = 2$. The above-mentioned initial values are suggested in [51].

The parameters' variation produces specific properties in the system, which can be used as a guideline to adjust the neuron's firing pattern: (i) the value of a controls the decay rate of u. (ii) The parameter b controls the sensitivity of u to changes in the membrane potential v below the threshold value. The integrator neuron has a low value of b which is why many spikes with a small time interval in between are needed to trigger a spike. (iii) The parameter c describes the value to which v is reset after firing. Therefore, decreasing its value enables to create spikes bursts since it makes the recovery to the original threshold value faster. (iv) The parameter d describes the reset in the value of u after a spike occurs, thus, lowering it creates a higher firing frequency for the same input current.

After setting the neuron parameters, the value for I^* and C_E can be concluded by analyzing the stability at v^* and u^* such that a specific firing rate at the output neurons is obtained for a certain input from multiple input neurons. I^* is even more critical to estimate in our study, as selective disinhibition has to be achieved and the neuron has to be maintained at the verge of firing to avoid excessive firing [52]. This ensures that only the correct motor neurons fire upon excitation of the corresponding sensory neurons.

To obtain I^*, we first equate $g(v^*, u^*) = a(bv^* - u^*) = 0$, and solve for $u^* = bv^*$. Then, substitute u^* into $f(v^*, u^*) = 0$ such that $0.04v^{*2} + (5 - b)v^* + 140 + I = 0$. The intersection of the u and v nullclines at one point is when the neuron starts to give continuous spikes, which means there is only one solution for the quadratic equation. The equilibrium points are given by $v^* = -(5 - b)/(2 \times 0.04)$ and $u^* = bv^*$. The value of I^* is finally computed by solving $f(v^*, u^*) = 0$. The parameter C_E must be chosen such that at the end of the training phase the selected firing behavior is still maintained. After setting C_E, the spiking behavior is tested. The chosen values for the motor neurons must not allow evoking spikes at low spiking frequency from sensory neurons. Note that increasing the frequency makes the learning process more susceptible to noise. Therefore, the motor neuron parameters are to be modified instead. The parameter b is then incremented slightly until a satisfactory performance is obtained.

Depending on the size of the data set and the number of neurons in each assembly, the number of iterations Itr* must be defined to build the map, which in turn determines the value of the learning rate gain S. If Itr* is relatively small, it will lead to a large S, which often leads to instability and noise sensitivity. A large Itr* results in an exhaustive (computationally demanding) training process.

To quantify the accuracy of the computed differential map approximated by the network, we define the following metric for the accuracy of estimation as introduced in algorithm 1:

$$\xi = \frac{1}{N}\sum_{n=1}^{N}\left|\arccos\left(\frac{\vec{v}_d \cdot \vec{v}_{est}}{\|\vec{v}_d\|\|\vec{v}_{est}\|}\right)\right| \qquad (17)$$

which is simply the difference between the desired spatial velocity $\vec{v}_d$ and the spatial velocity $\vec{v}_{est}$ obtained upon execution of the estimated motor command $\dot{\theta}_{cmd}$ as shown in figure 6 (calculated from the decoding equation (8)); the scalar $N > 0$ is the number of points in the workspace over which the difference is measured to obtain a mean error value. The tuning

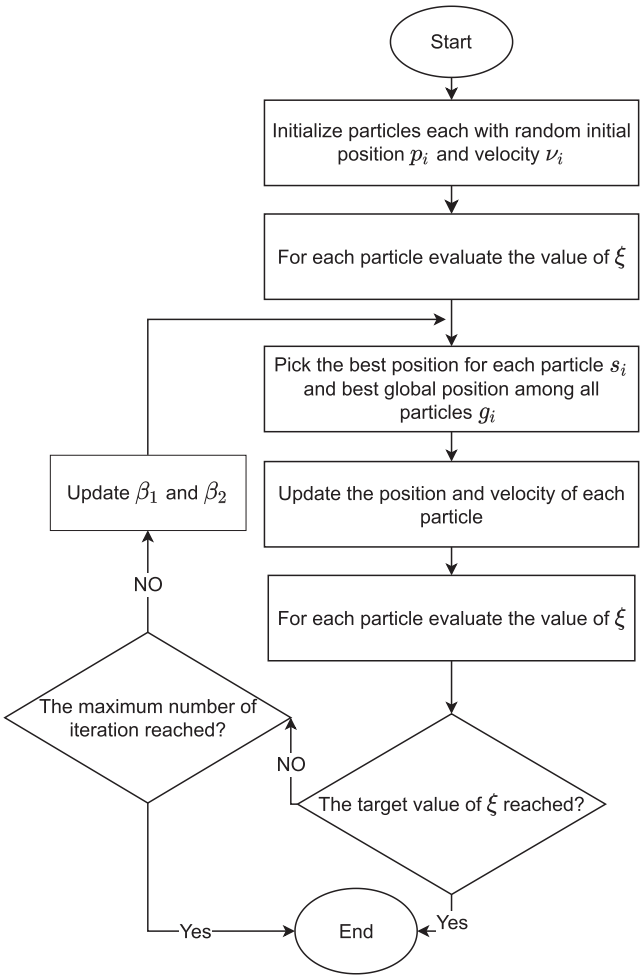


Figure 7. A flowchart for the tuning of the parameters carried out using PSO.

process continues until the value of ξ is below some threshold value e_{th}, indicating that the network's performance is acceptable.

2.6. Automated tuning of the network parameters

While the previous subsection introduces a guideline for a possible choice of good values for the network parameters to have a good performance, the high dimensional space for these parameters makes it difficult to obtain optimum performance with only trial and error. PSO is among the metaheuristics used to look for optimum value for a defined search space. While a global minimum is not guaranteed, but with an appropriate objective/fitness function, convergence towards a satisfying solution is achieved upon reaching a threshold value. The PSO acts to minimize the objective function, which is chosen as equation (17), to give good candidate solutions that minimize the mean error while reaching the targets. With a big enough population size and number of iterations/generations (relative to the number of parameters to be optimized in the search space), such an optimal solution/particle can be reached. To narrow down the limits of the search space, the guide from the previous subsection makes it easier to define the range of values for the parameters to be optimized to reduce the size of population and number of iterations needed. In the PSO, as shown in figure 7, each particle i in the population has a position p_i and a velocity ν_i while moving to search for an adequate solution. For $n_{\mathcal{P}}$ is the number of parameters $\mathcal{P}$ in the search space, $p_i, \nu_i \in \mathbb{R}^{n_{\mathcal{P}}}$. Initially, both the position and velocity of each particle are randomly initialized within a predefined range. Then the fitness of each member in the population is calculated. After each iteration, p_i is updated such that:

$$p_i(t+1) = p_i(t) + \nu_i(t). \tag{18}$$

Thus, the velocity ν_i is affected by the best local solution found by the particle $s_i \in \mathbb{R}^{n_{\mathcal{P}}}$ and the best global (found by the whole population) solution $g_i \in \mathbb{R}^{n_{\mathcal{P}}}$, such that:

$$\begin{aligned}\nu_i(t+1) = \nu_i(t) &+ \beta_1 r_1(s_i(t) - p_i(t)) \\ &+ \beta_2 r_2(g_i(t) - p_i(t))\end{aligned} \tag{19}$$

where β_1 and β_2 are the acceleration coefficients for the best local and global solutions, respectively. Both r_1 and r_2 are random values ranging from 0 to 1.

Table 1. PSO search space.

	Upper and lower search limits					
	$\mathcal{P}$					
$\mathcal{L}$	a_m	b_m	c_m	d_m	A_s	A_m
Min	0.02	−0.4	−70	2	5	5
Max	0.5	0.25	−55	8	80	80

Thus, the velocity of the particle depends on its velocity in the previous iteration and its position from the best local and global solutions. The two coefficients β_1 and β_2 decrease linearly as the search proceeds. The maximum and minimum values of β_1 and β_2 are presented in section 4. The parameters chosen for this study are the motor neurons parameters (a_m, b_m, c_m and d_m) and the amplitude of the training phase input current to the sensory neurons A_s and motor neurons A_m defined in equation (4). Thus, $\mathcal{P} = (a_m, b_m, c_m, d_m, A_s, A_m)$ and $n_\mathcal{P} = 6$ for the studied case. The upper and lower limits for the search space are defined in table 1 as $\mathcal{L}_{min}$ and $\mathcal{L}_{max}$, respectively. The maximum and minimum velocities are set initially to be 15% of the searching range and keep decreasing to reach only 5% by the last iteration/generation.

3. Simulation results

3.1. Network layout selection

To test the proposed network and tuning method, the summation of two variables ($n_1 + n_2 = n_{sum}$) is carried out in different approaches. In figure 8(a), the sum is estimated using two 1D-layers that encode two variables ('n_1' and 'n_2') connected through all plastic connections to a 1D-layer encoding the result ('n_{sum}'). Random numbers are generated within a range and introduced to the input layer, with their summation introduced into the output layer. The network parameters are tuned as described in section 2.6, such that during the training phase, firings in both input and output layers allow the plastic connections to represent the desired summation function. After training, the sum is estimated from the decoded values at the output layer. The approach gives a mean error of 7.5%.

Figure 8(b) shows another network layout used to test our method. A 2D input layer is used instead of two separate 1D layers. The difference from the previous layout is that the neurons' activity is estimated by multiplying the normally distributed activity of both variables. The mean error for this layout is around 2%. To build the proposed DMSNN, we use the former approach. The rationale behind this choice is elaborated in section 5.

3.2. 2DOF planar robot

To verify the effectiveness of our method for visual servoing tasks, a simulation model of a 2DOF planar robot with revolute joints is built. The differential kinematics of this system are:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}-l_1\ \sin(\theta_1) - l_2\ \sin(\theta_{12}) & -l_2\ \sin(\theta_{12})\\ l_1\ \cos(\theta_1) + l_2\ \cos(\theta_{12}) & l_2\ \cos(\theta_{12})\end{bmatrix} \begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix} \tag{20}$$

where l_i denotes the link length, and $\theta_{12} = \theta_1 + \theta_2$. With the above definition of the Jacobian matrix, we can derive the inverse differential mapping (based on equation (2)), which is then used to generate training data for the SNN. During the training process, normalized spatial velocity vectors are fed at random joint angles to the corresponding assemblies in the input layer along with the corresponding angular velocities to the assemblies of the output layer. The firing activity of the neurons is monitored, as shown in figure 9, to make sure the network parameters are correctly set to obtain the desired firing frequencies and patterns of the neurons. For the accuracy of the robot simulation, around the singular configurations, a small value (10^{-5}) is added to the determinant of the Jacobian matrix to avoid the unbounded values of joint velocities.

After training, the network is tested by providing a random target in Cartesian workspace x_d for a random initial position x_i. By generating a normalized spatial velocity vector $\vec{v}_d$ from the current position x to reach the target, the estimated angular velocities ($\dot{\theta}_{cmd}$) at each joint can be decoded from the assemblies of the output layer.

To quantify the performance, we calculate the minimum distance δ from the current position to the line segment $\overline{x_i x_d}$ as shown in figure 6. The target path is a straight line, thus, the shortest distance at each point is defined by the normal to $\overline{x_i x_d}$. However, in other cases, a more complex path may be required. For a robot moving along the path C, divided into N_C points, given a reference target path Ω, composed of N_Ω points, the mean error e_{mean} is calculated by averaging the maximum deviation δ_{max} over N_{trials} by applying algorithm 2. The training is done over approximately 3000 iterations and tested over 5 times repetition of servoing to 15 different targets (generated randomly in the workspace), to obtain the number of successful trials (in which the final end-effector error e_x is below the threshold value, that is chosen to

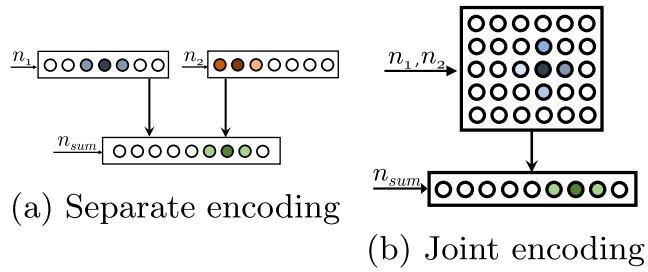


Figure 8. A schematic of the two possible ways to perform summation of two variables (n_1 and n_2). In this paper, (a) is the chosen approach.

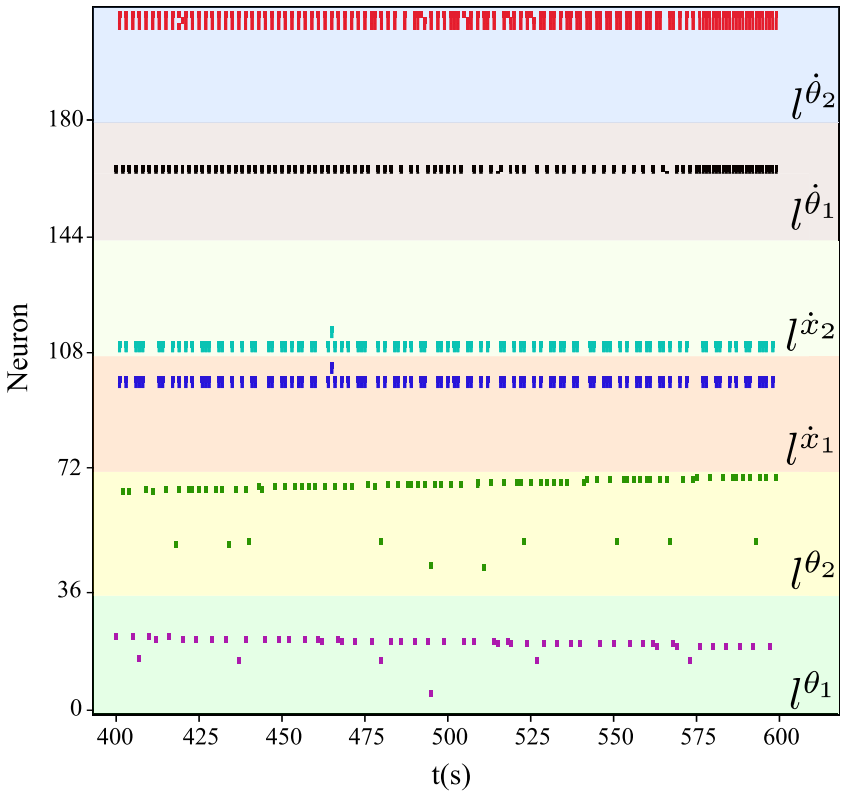


Figure 9. A spike raster plot of neurons in all assemblies in the network during the training phase.

be 1 mm here) and the standard deviation of the maximum deviation $\delta_{\max}$ for approaching chosen targets is given by:

$$\sigma_{\text{servoing}} = \sqrt{\frac{\sum_{n=1}^{N_{\text{trials}}} (\delta_{\max}(n) - e_{\text{mean}})^2}{N_{\text{trials}} - 1}}. \quad (21)$$

Figure 10(a) shows the plot of the percentage of successful trials for the chosen parameters against the number of training iterations. It can be concluded that the learning, in that case, reaches a stable state after around 2700 iterations. Moreover, the figure includes the plot of the network performance with a slight change of the key parameters (in this case parameter b for the output neurons as discussed earlier) with a noticeable degradation in the learning ability and stability of the network. Thus, it can be concluded the importance of fine-tuning the network parameters. Additionally, figure 10(b) shows the development of the performance of servoing at different points in the workspace. It shall be noted that the performance of the network is degraded in some areas while the training proceeds, thanks to the generalization of the learning process. Thus, instead of training in a local area, the whole defined workspace is instead learnt. This means that initially specific examples are learnt and all neurons act to represent these examples, but as the training proceeds, the representation includes a wider set of examples.

4. Experiments

4.1. Experimental setup

The proposed SNN is simulated using NeMo library, a tool developed to simulate SNN [53, 54]. A UR3 robot is set with an Intel RealSense D415 camera in

Algorithm 2. Calculating mean error.

```
for i = 1 to N_trials do
    e_i = 0
    for j = 1 to N_C do
        e_j = || C(j) − Ω(1) ||
        for k = 2 to N_Ω do
            e_k = || C(j) − Ω(j) ||
            if e_k ⩽ e_j then
                e_j = e_k
            end if
        end for
        if e_j ⩾ e_i then
            e_i = e_j
        end if
    end for
    Stack error δ_max(i) ← e_i
end for
e_mean = (1/N_trials) Σ_{n=1}^{N_trials} δ_max(n)
```

the setup as shown in figure 11(a). The D415 is an RGB-D camera providing real-time color frames and depth maps. Image registration is carried out by aligning the color and depth frames, then the end-effector position can be measured through color filtering of the manipulated object. For effective color filtering, the image is blurred to remove high-frequency noise, then the color frame is converted to HSV color space for more robust performance independent of the light intensity.

A mask, in the desired color range, is applied to obtain a binary image, which is then subjected to some morphology operators to remove the small blobs. These operations allow to filter the image and make it easier to discriminate the colored end-effector as the biggest contour. Afterwards, the moment of this contour is calculated such that $M_{ij} = \sigma\phi(x_1, x_2){x_1}^{(i)}{x_2}^{(j)}$. Then, the centroid (ρ_x, ρ_y) is calculated in pixel units based on the moment such that $\rho_x = M_{10}/M_{00}$ and $\rho_y = M_{01}/M_{00}$, see [55].

The location at the center of the object is then converted to world coordinates by using the camera's intrinsic parameters [56]:

$$x_1 = (\rho_x - c_x)\frac{x_3}{F}, \qquad x_2 = (\rho_y - c_y)\frac{x_3}{F} \tag{22}$$

where ρ_x and ρ_y denote to the end-effector position in pixels, x_3 is the end-effector's depth, c_x and c_y denote the principal point and F is the focal length. The following first-order filter is used to remove noise from these visual measurements:

$$s_f(t+1) = s_f(t) - \lambda(s_f(t) - s(t+1)) \tag{23}$$

where s and s_f are the variable before and after filtering, respectively, and $\lambda > 0$ denotes the filter's gain. Similarly, the position of the target is converted from pixels (input given by mouse clicks on the camera display) to world coordinates based on the depth that is defined and can be incremented or decremented by button clicks. The target and end-effector positions are compared every iteration to update the error e_x and thus the desired end-effector velocity v_d as identified in equation (7).

To train the network, the robot is driven linearly in the joint space between randomly generated angles. UR script-based programming allows the execution of such a motion in the joint space by defining only the desired positions and speeds of joints. Data is collected at a constant sampling rate of 25 Hz, then filtered. As mentioned earlier, κ_θ varied within the range [0.03, 0.1] rad s^{-1}, and κ_x varies depending on the maximum and minimum Cartesian velocities recorded during the babbling. Figure 12(c) shows the positions obtained from the camera after filtering and those obtained using the internal robot sensors. Each neuron assembly is fed with the corresponding data to let the plastic connections develop to form the required differential mapping. Once the training phase ends (dependent upon the number of neurons and the size of the workspace), the strength of the synaptic connections is kept constant. Random targets are given across the robot workspace as shown in figures 12(a) and (b). The current angular position and the desired spatial velocity are updated every 20 ms.

4.2. 3DOF planar robot

For this case, three joints are controlled to guide the UR3 robot in planar motion, as shown in figure 13. The network consists of five input assemblies ($l^{\theta_{1:3}}$ and $l^{\dot{x}_{1:2}}$) and three output assemblies ($l^{\dot{\theta}_{1:3}}$). The network parameters, as well as the number of neurons in each assembly, are shown in tables 2 and 3, for manual and PSO tuning, respectively. The parameters of the output neurons chosen by the PSO are close to those of an integrator neuron to validate the initial assumptions as discussed in subsection 2.6. The three joints have ranges of: $[-180°, -90°]$, $[-45°, 0°]$ and $[90°, 180°]$.

From the motor babbling data collected, the maximum and minimum values for both the spatial and angular values are obtained. For each layer with $\mathcal{N}_3$ neurons, the central value ψ_c encoded by each neuron is assigned by dividing the range of each variable

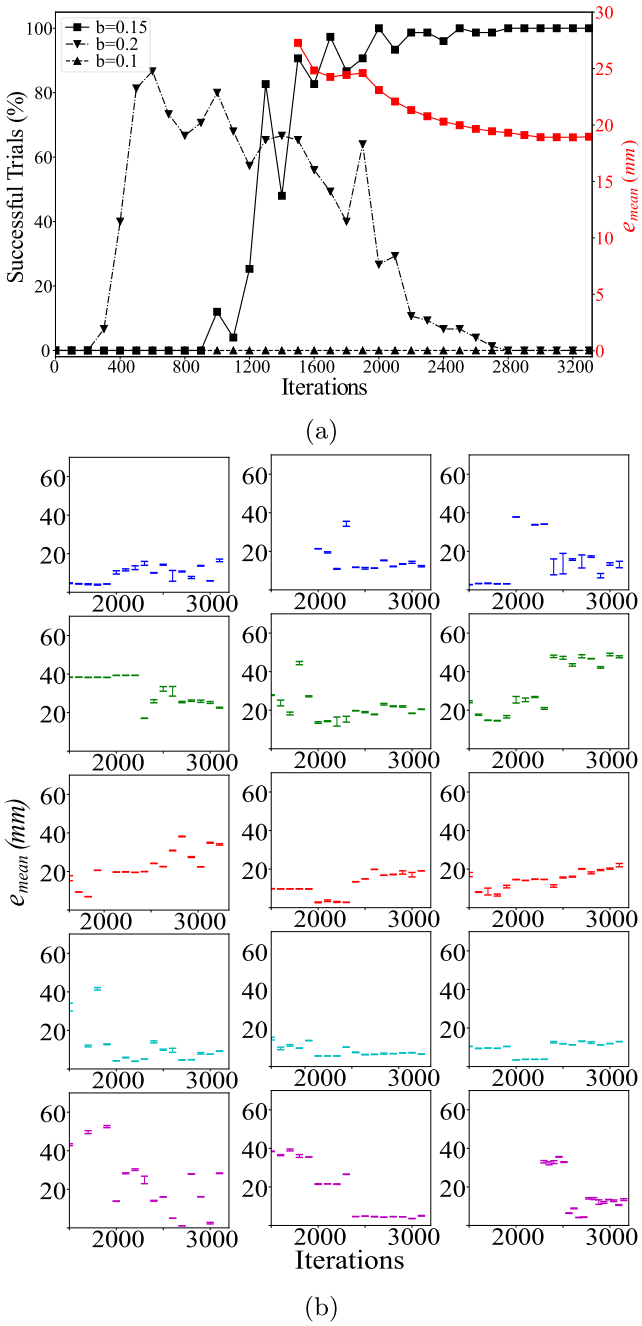
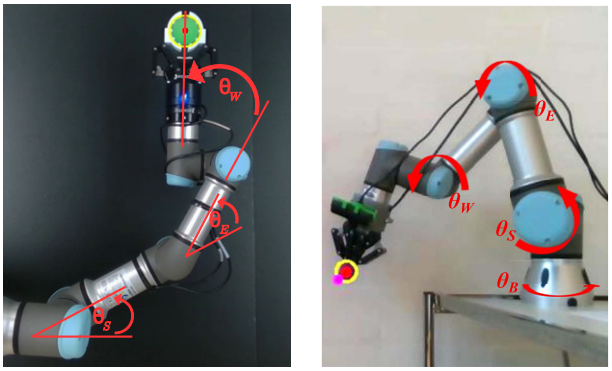


Figure 10. (a) Plot of both the number of successful trials (in black), and the mean error (in red) during the control phase versus the number of training iterations. The solid, dotted and dashed black lines are related to the output neurons with the parameter $b = 0.15, 0.1$, and 0.2, respectively. The mean error is only plotted for the case in which successful trials are above 80%. (b) Plot of the mean error and standard deviation for each individual target approach (unsuccessful trials are discarded from calculations and plot).

evenly over the whole layer. The update of the synaptic strength is shown as heatmaps in figures 14(a) and (b) for manually tuned parameters, and figures 15(a) and (b) for parameters tuned using PSO. Each heatmap depicts the relation between one assembly from the sensory layer to one assembly from the motor layer, where each pixel gives the strength of an excitatory synapse connecting one sensory neuron to one motor neuron. After running the simulation for 6000 iterations, the strength of synapses between a motor layer to the sensory layer is modulated to represent the required differential map. The robot is then given 15 random points to reach with the end-effector through a visual servoing process. The reaching is considered successful if the end-effector position is less than 3 pixels away from the target in each coordinate. As shown in figure 13, a target point is provided and the robot automatically moves towards it by using the decoded motion command $\dot{\theta}_{\text{cmd}}$. The speed of motion varies by changing the value of κ_x, where in this case $\kappa_x \in [0.8, 1.2]$ cm s^{-1}. The distance between the end-effector and the target is represented as the norm $\|e_x\| = \|x - x_d\|$. Figure 16(a) shows the mean error for the feedforward SNN (i.e., without interinhibitory



(a) UR3 Planar configuration (b) UR3 Spatial configuration

Figure 11. The UR3 moves (a) planar motion while controlling 3DOFs and (b) spatial motion while controlling 4DOFs.

connections), and both the manually and automatically tuned DMSNNs, where the 3 networks could successfully reach the 15 targets. However, it is evident that an improvement in performance, as well as a great reduction in the number of neurons per assembly, are achieved after applying the PSO and using the obtained parameters. The improvement occurs as a decrease in the mean error (defined as maximum deviation from the reference path) and an increase in the mean speed of the end-effector to approach the reference velocity. Hence, less time is needed to reach the target and faster servoing is achieved.

4.3. 4-DOF spatial robot

In this case, four joints are controlled to guide the UR3 robot as shown in figure 11(b).

The network consists of seven input assemblies ($I^{\theta_{1:4}}$ and $I^{\dot{x}_{1:3}}$), and four output assemblies ($I^{\dot{\theta}_{1:4}}$), each assembly consisting of $\mathcal{N}_4$ neurons. The network parameters and the number of neurons in each layer are shown in tables 2 and 3, for manual and PSO tuning, respectively. It shall be noted that the manually tuned neuron parameters for both cases of 3 and 4 DOFs are the same, as changing only the values of A_s and A_m allows to sustain the controlled firing and learning process. The four joints ranges are: $[-200°, -170°]$, $[-75°, -45°]$, $[90°, 110°]$ and $[-200°, -160°]$. The heatmaps depicting the update of weights for manually tuned parameters at the 4000 and 9000 iterations are shown in figures 14(c) and (d), respectively. Figures 15(c) and (d) depicts the update of weights for parameters tuned using PSO at 4000 and 9000 iterations, respectively. It is clear from the comparison of figures 14 and 15 that the synaptic weights in the latter are distributed over the whole map. Hence, the tuning of parameters using PSO leads to a better contribution of all neurons in the network to form the desired map, and thus reducing the number of required neurons. Similar to the previous setup, randomly chosen 20 end-effector targets across the workspace are given to the robot. The target reaching, in this case, is considered successful when the end-effector is less than 3 pixels away from the target in x and y coordinates and 5 mm in depth (z coordinate). The feedforward SNN reaches successfully to 16 out of the 20 targets, while both the manually and automatically tuned DMSNNs successfully reached the 20 targets with the mean error as shown in figure 16(a) and the mean velocities as shown in figure 16(b). The reference velocity in this case changes by changing κ_x, where $\kappa_x \in [0.2, 0.5]$ cm s^{-1} which is lower than that of the 3DOF case to allow the network to average the output over more iterations and negate the effect of the increase in error due to the noisy depth readings. Representative results for these 4-DOF visual servoing tasks are depicted in figure 17 before and after optimizing the network parameters via PSO. It shall be noted that the ripples in the plot are due to the noise in the data collected by the depth sensor, which can be judged by comparison to the plots of velocities based on the readings collected by the motor encoders as shown in figure 12(d). The accompanying multimedia video (https://stacks.iop.org/BB/16/036008/mmedia) demonstrates the performance of our new method with many experimental results.

5. Discussion

As shown in section 3.1, the accuracy of the summation of two numbers by the network in the case of multiple one-dimensional arrays is lower than that obtained in the case of a multi-dimensional array. However, this result comes at the expense of the network size where the 1D layers require only $2\mathcal{N}$ neurons, for $\mathcal{N}$ is the number of neurons per layer, while the 2D layer needs $\mathcal{N}^2$ neurons. This means that for an O-dimensional case, $\mathcal{N}^O$ neurons would be needed. However, note that the explicit use of visual feedback in our formulation provides a valuable *rectifying* property to the network (i.e., it corrects for small errors in an otherwise open-loop controller) while demanding moderate computational power.

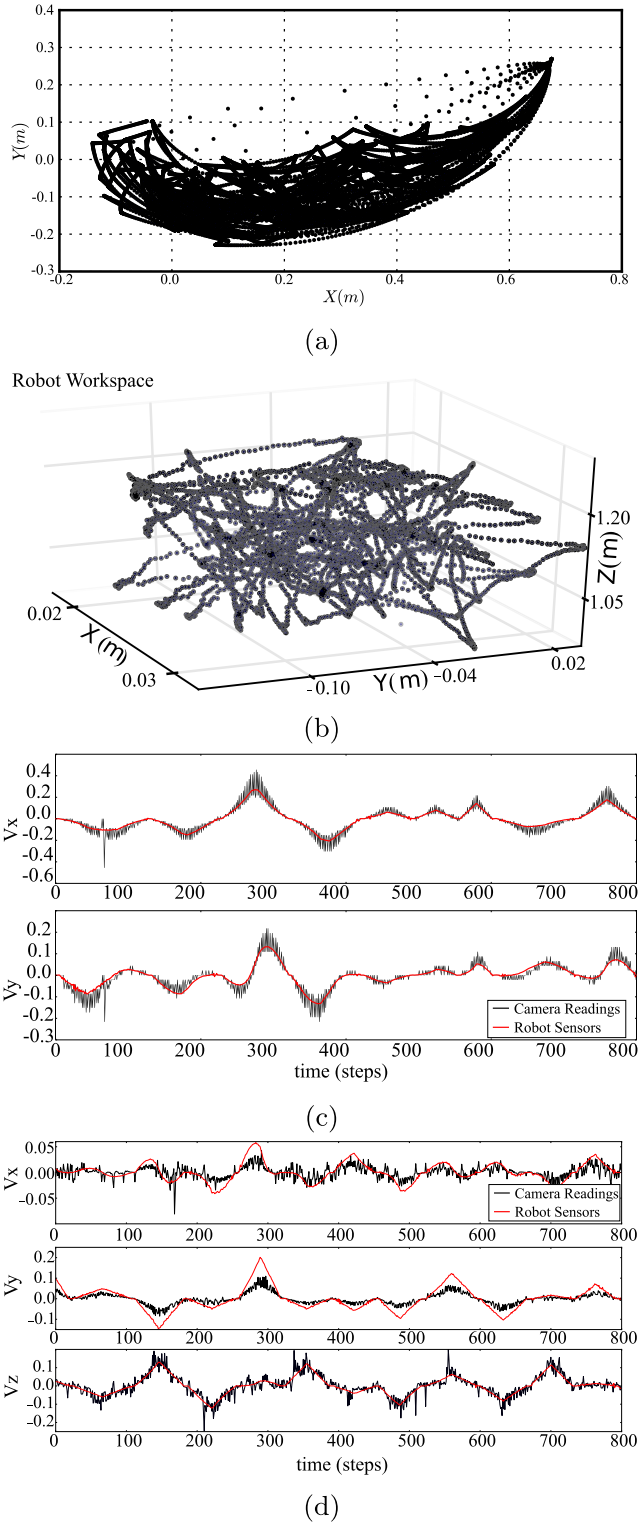


Figure 12. Plot of data collected for training the DMSNN for (a) 3DOFs and (b) 4DOFs, and velocities as measured from the camera (after filtering) against that estimated by the internal sensors of the robot for (c) 3DOFs and (d) 4DOFs.

Moreover, in this study, the SNN is used to guide the robot for visual servoing, where it requires fast state updates, hence the proposed network is to be used instead as it compromises between real-time operation and sufficient accuracy. However, in the case of accurate mapping of robot states and commands, the second approach is to be used, or additional hidden layers are to be added to the network for higher accuracy and better estimates. Another feature, which is not evident from the summation test studied, is that SNN better approximates real-life operations where the values to be encoded/decoded are continuous, i.e. with no discontinuity or jumps. For the neurons in the network to evoke a spike, it needs first to let the membrane potential build up over a series of time steps. While incorporating the Gaussian

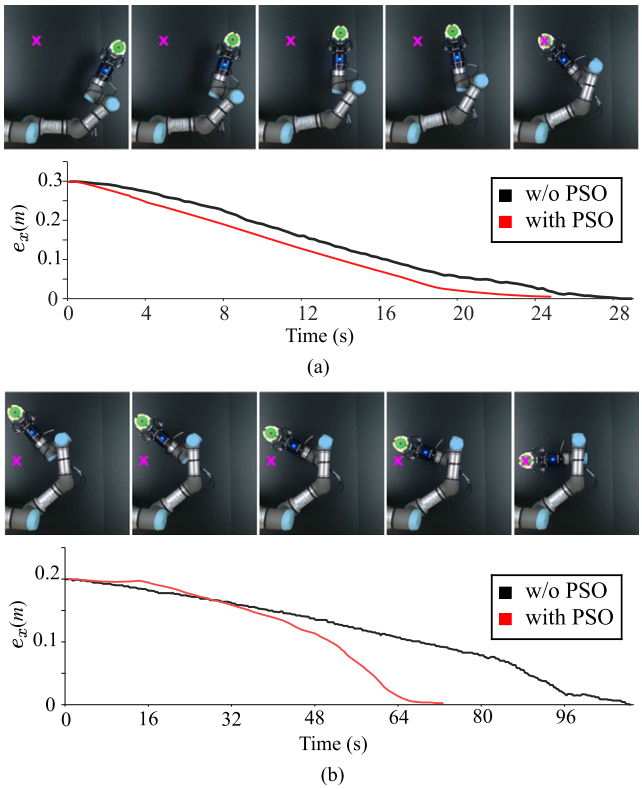


Figure 13. The UR3 while performing planar motions to 2 different targets from different initial positions in (a) and (b) with the norm of the error e_x plotted during motion before and after optimizing the network parameters using PSO.

Table 2. Manual tuning network parameters.

Neuronal layers						
	a	b	c	d	$A_{3/4}$	$\mathcal{N}_{3/4}$
I^{θ_i}	0.1	0.2	−65	2	20/15	68/136
I^{x_i}	0.1	0.2	−65	2	20/15	68/136
I^{ϕ_i}	0.02	0.15	−55	6	6/6	68/136

Synaptic connections				
	Param			
DOF	S	τ_1/τ_2	C_I/C_E	Itr*
3 (planar)	0.05	20/18	−4/4	6000
4 (spatial)	0.03	20/18	−5/5	9000

Table 3. PSO tuning network parameters.

	a	b	c	d	A_s	A_m	$\mathcal{N}$
3D neuron parameters							
I^{ϕ_i}	0.22	−0.25	−68	6.8	30	40	17
4D neuron parameters							
I^{ϕ_i}	0.11	−0.24	−68	7.8	27	50	27

activity, this leads to sharing excitation with neurons in the neighborhood, and it becomes more likely for the adjacent neurons to be triggered in the next cycles as well. As the adjacent neurons encode values close to each other, this achieves a continuous smooth operation and avoids sudden changes and jerky motions. So, despite the fact that in the asynchronous SNN, as in our case, neurons that release a spike first tend to spike again, the fine-tuning of the parameters acts to facilitate more accurate approximations and

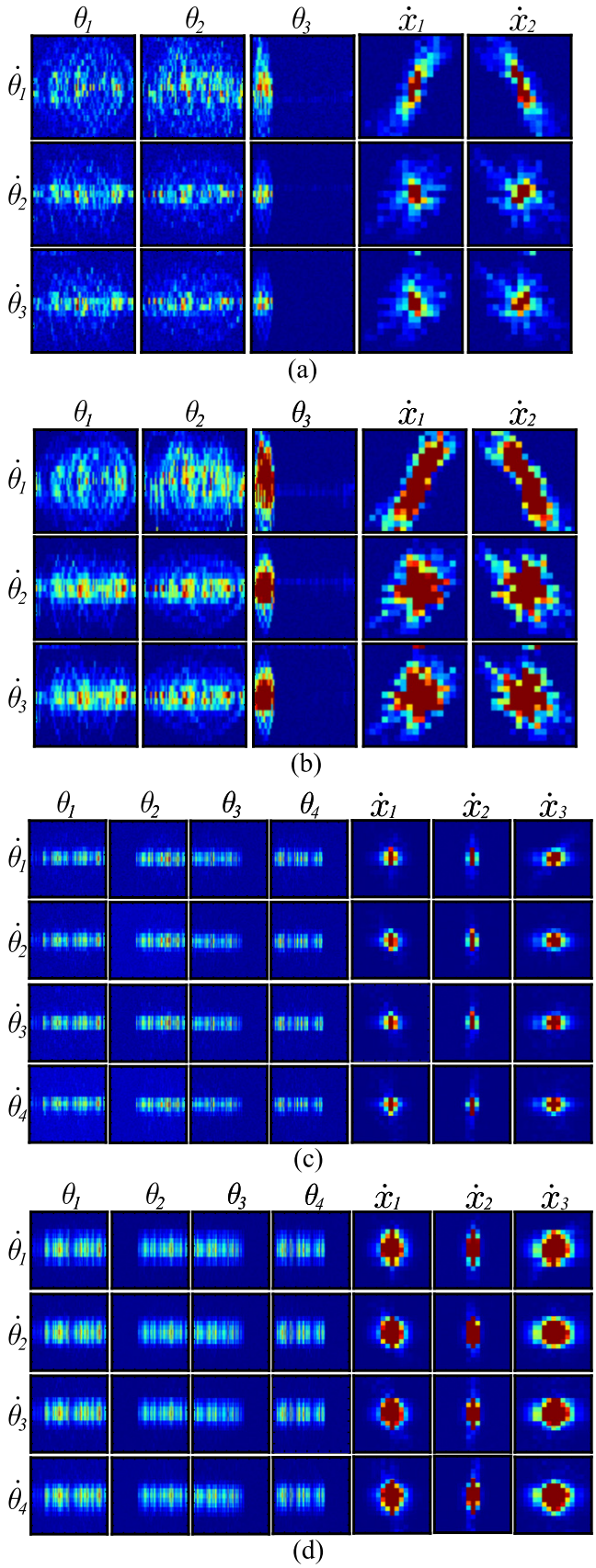


Figure 14. Heatmap of the weights update process for the 3DOF case of study at (a) 3000 and (b) 6000 iterations, and for 4DOF case at (c) 4000 and (d) 9000 iterations with manually tuned parameters. The dark blue color corresponds to zero weight while the dark red color corresponds to the maximum weight.

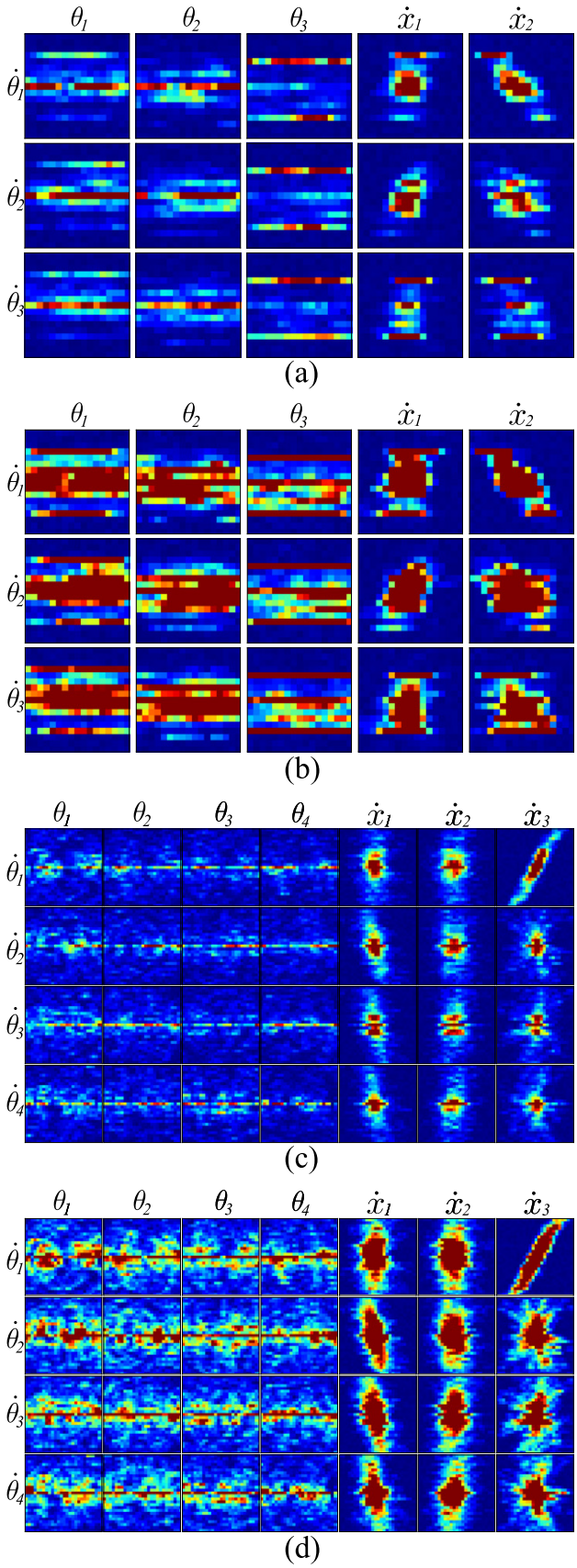


Figure 15. Heatmap of the weights update process for the 3DOF case of study at (a) 3000 and (b) 6000 iterations, and for 4DOF case at (c) 4000 and (d) 9000 iterations with parameters tuned via PSO.

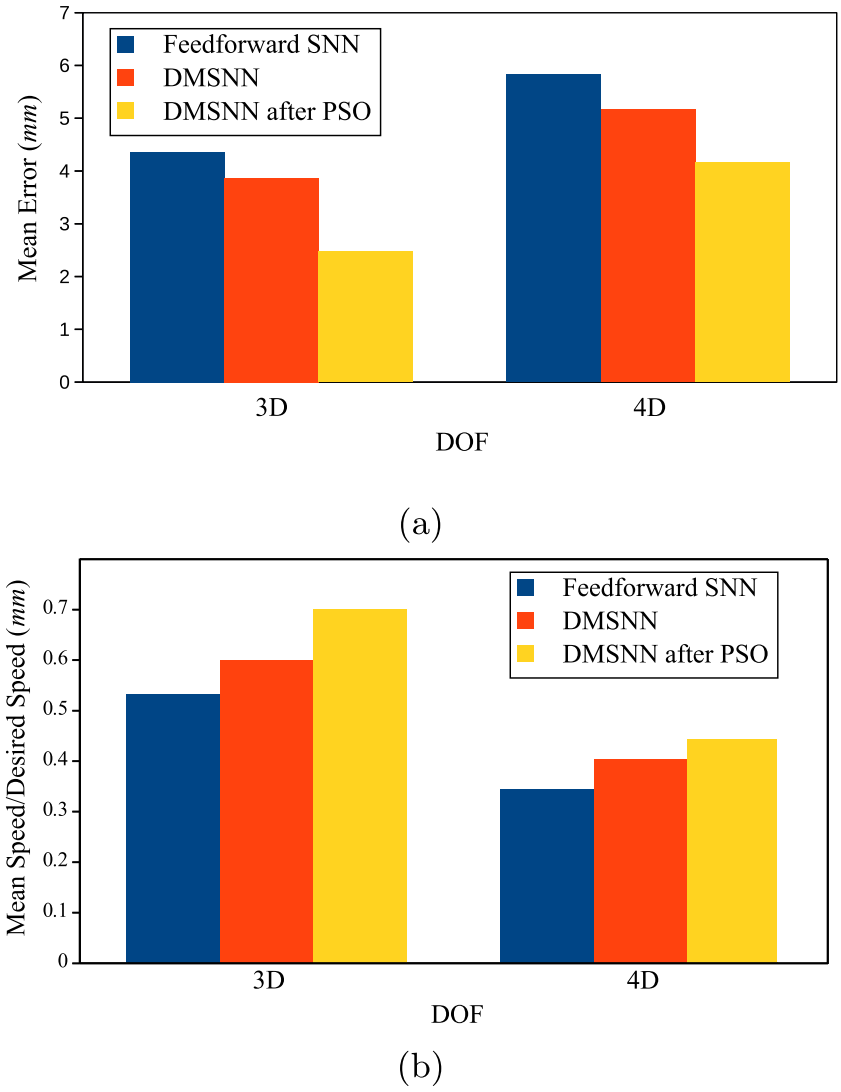


Figure 16. (a) The mean error and (b) mean speed for reaching several random targets in both 3 and 4 DOFs cases.

the activity of the neighboring neurons acts to correct the estimations during the servoing process.

The proposed network relies on biologically plausible models of neurons and synapses, which are more complex compared to other relatively simpler models, to both preserve features of the biological nervous system as much as possible and serve as a future building block for simulating the whole hierarchy of the biological motor system. An example of this is the simple Izhikevich neuron model [37] used in the proposed network, instead of the commonly used LIF [36]. This work also exploits the potential of such SNN, following some procedures for fine-tuning of the network parameters as described in subsection 2.6, where only two layers (input and output) and a low number of neurons are needed to develop a differential map for multi-DOF robots. Additionally, the noisy data is well-handled thanks to the inhibition that occurs for the distal firing of pre and postsynaptic neurons, which weakens the synapses connecting uncorrelated neurons, aside from the inter-inhibitory connections that inhibit distal neurons to avoid undesired firing.

The developed network can then be compared to the work from [25], where an SNN is used to build a map for the static transformation of coordinates for a simulated 2DOF robot (let us denote it as CTSNN), and the work from [57], where an SNN modeling the sensorimotor cortex is used to guide a simulated 2DOF robot in a reaching task (and denoted as SMCSNN). Both these studies target to build a map that can be used to guide a simulated 2DOF robot which is similar to the studied case thus providing a valid comparison. Table 4 highlights the differences in terms of the number of training iterations (Itr^*) and time ($T^*(s)$), the network size ($\mathcal{N}_{\mathrm{net}}$) and the final error e_x.

It can be concluded that both the simulation time and the real time required for training and running the network is less for the proposed network, thanks to the fine-tuning of the network parameters and maximizing the benefit from the spiking nature of the network.

Following the proposed tuning method, in the case of the spatial configuration, each neuron

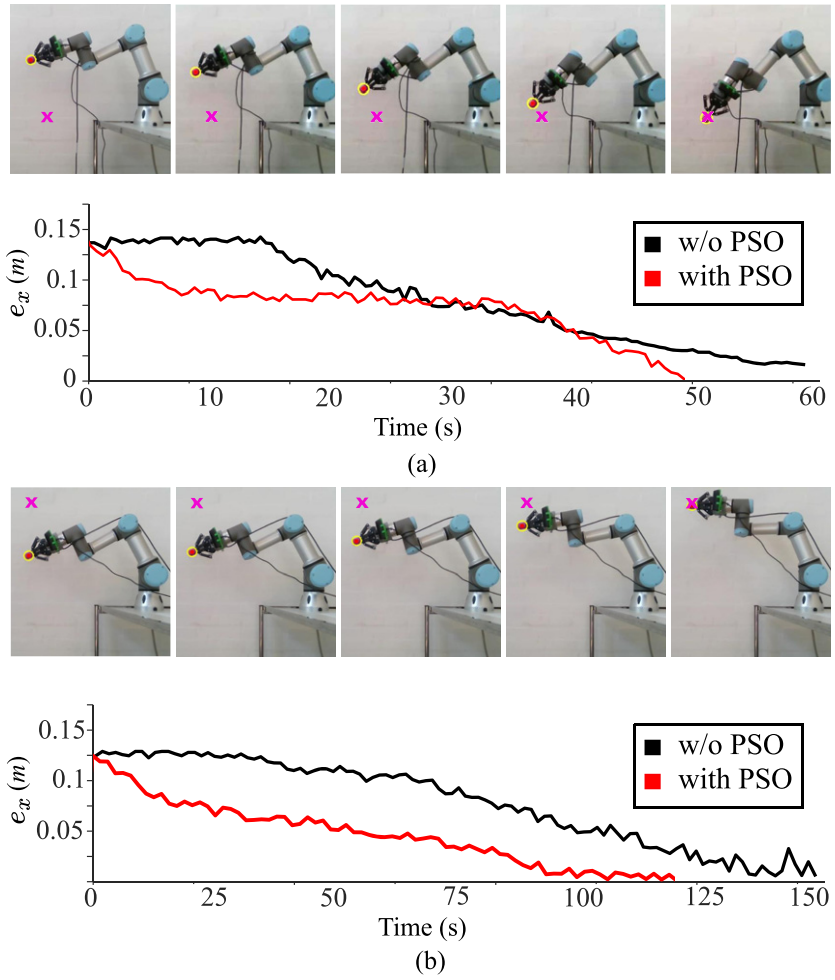


Figure 17. The UR3 while performing spatial motions in (a) and (b) with the norm of the error in the 3 axes e_x plotted during motion before and after optimizing the network parameters using PSO.

Table 4. Networks comparison.

	Param			
Net	Itr[a]	T^{a}(s)	e_x	$\mathcal{N}_{\text{net}}$
CTSNN	≈4000	800	NA	2000
SMCSNN	200*ep*	3000	≈1 mm	704
DMSNN	3000	30	⩽1 mm	216

[a.] The number of iterations for CTSNN is estimated based on 8000 s training divided into 1600 time steps (each time step 0.125 ms) for each stimulus introduced. The number of training steps is not mentioned for the SMCSNN, instead, the number of epochs is mentioned (which in this case is the motion from one starting point to a target point) and each epoch takes around 15 s. Each iteration in the DMSNN is 80 ms of simulation time (which is only 10 ms of real time), through which a certain stimulus is introduced to the network.

assembly consists of around 136 neurons and 27 neurons only after tuning the parameters using PSO, compared to around 1000 neurons in [26]. Moreover, the study in [26] suggests to divide the input into separate bins to improve the learning, but both the formulation and the explanation of how this is achieved is lacking, and the method to set and tune the network parameters is not mentioned.

In [20] each assembly is constructed of around 100 neurons only, however, a hidden layer is needed which increases the size of the network and complexity of the tuning process. Moreover, the robot in this work needs to approach only 100 points, and the data recorded while moving to these target points is sufficient for the training process, compared to 6426 target points in [23]. The robot succeeds in all trials to reach the destined targets through visual servoing with the reduction in the mean error through the modified architecture and the automatically tuned network parameters.

The time required to reach the target varies depending on the distance from the current position and the richness of data trials available along the path between the current and target pose. This can be seen in the two examples in figure 13, where in figure 13(a) the robot takes less time to cover a bigger distance compared to figure 13(b). This can be explained by having the training data collected from motor babbling in joint space with more examples of the robot moving in waving-like trajectories and less abundant data for linear motion in task space. Similarly, for the trial in figure 17(a), less time is needed compared to the trial in figure 17(b) as the target in the latter is close to the boundary of the studied section of the workspace which has less training examples. In these previous examples, the DMSNN-based servoing has less deviation from the reference path, and thus takes less time after employing the parameters generated by the PSO. Richer data collected from motions in both joint and task space can be used in a future study to role out the effect of varying the data on different motion types, and the possibility of emerging of motion primitives from such behavior.

It can be noticed from figures 12(c) and (d), the training data is filtered using only first-order filter and contains more noise in case of the spatial motion, due to the noise in depth readings, however, the DMSNN is still able to approximate the differential relationship and build the desired map. This can be justified by the depression in the strength of uncorrelated neurons due to the chosen STDP rule.

6. Conclusions

Utilizing both the proposed architecture and the tuning guideline, the DMSNN provides a way to build a differential map to drive robots with many DOFs through multimodal servoing tasks. The network is featured by real-time operation and a small amount of data, compared to similar methods in the literature, is needed for training. The current limitations of this method are related to the limited resolution of the network output. Future work will focus on improving the output resolution, as well as deriving a mathematical formula to conclude the optimal parameters for neuron firing and STDP learning. We also plan to use this method for conducting vision-guided shape control tasks with deformable objects, as in [58], and apply a cerebellar controller to enhance the performance and reduce the deviation from the path, as in [22].

Acknowledgments

This research work is supported in part by the Research Grants Council (RGC) of Hong Kong under Grant No. 14203917, in part by PROCORE-France/Hong Kong Joint Research Scheme sponsored by the RGC and the Consulate General of France in Hong Kong under Grant F-PolyU503/18, in part by the Chinese National Engineering Research Centre for Steel Construction (Hong Kong Branch) at PolyU under Grant BBV8, in part by the Key-Area Research and Development Program of Guangdong Province 2020 under project 76 and in part by The Hong Kong Polytechnic University under Grant G-YBYT and 4-ZZHJ.

Data availability statement

The data generated and/or analyzed during the current study are not publicly available for legal/ethical reasons but are available from the corresponding author on reasonable request.

ORCID iDs

Omar Zahra https://orcid.org/0000-0003-1644-6480
Silvia Tolu https://orcid.org/0000-0003-1825-8440
David Navarro-Alarcon https://orcid.org/0000-0002-3426-6638

References

[1] Navarro-Alarcon D, Cherubini A and Li X 2019 *On Model Adaptation for Sensorimotor Control of Robots, in 2019 Chinese Control Conf. (CCC)* (IEEE) pp 2548–52
[2] Fagard J, Esseily R, Jacquey L, O'Regan K and Somogyi E 2018 Fetal origin of sensorimotor behavior *Front. Neurorob.* **12** 23
[3] Aoki T, Nakamura T and Nagai T 2016 Learning of motor control from motor babbling *IFAC-PapersOnLine* **49** 154–8
[4] Xiong X, Wörgötter F and Manoonpong P 2015 Adaptive and energy efficient walking in a hexapod robot under neuromechanical control and sensorimotor learning *IEEE Trans. Cybern.* **46** 2521–34
[5] Pinto L and Gupta A 2016 Supersizing self-supervision: learning to grasp from 50K tries and 700 robot hours *2016 IEEE Int. Conf. On Robotics and Automation (ICRA)* (IEEE) pp 3406–13
[6] Pierson H A and Gashler M S 2017 Deep learning in robotics: a review of recent research *Adv. Robot.* **31** 821–35
[7] Arulkumaran K, Deisenroth M P, Brundage M and Bharath A A 2017 Deep reinforcement learning: a brief survey *IEEE Signal Process. Mag.* **34** 26–38
[8] McCulloch W S and Pitts W 1943 A logical calculus of the ideas immanent in nervous activity *Bull. Math. Biophys.* **5** 115–33
[9] Widrow B and Stearns S D 1985 *Adaptive Signal Processing* (Englewood Cliffs, NJ: Prentice-Hall)
[10] Maass W 1997 Networks of spiking neurons: the third generation of neural network models *Neural Netw.* **10** 1659–71
[11] Chittka L and Niven J 2009 Are bigger brains better? *Curr. Biol.* **19** R995–R1008
[12] Makino H, Hwang E J, Hedrick N G and Komiyama T 2016 Circuit mechanisms of sensorimotor learning *Neuron* **92** 705–21
[13] Thorpe S, Fize D and Marlot C 1996 Speed of processing in the human visual system *Nature* **381** 520–2

[14] Wang G and Pavel M 2005 A spiking neuron representation of auditory signals *Proc. of the Int. Joint Conf. on Neural Networks 2005* vol 1 (IEEE) pp 416–21

[15] Krichmar J L 2018 Neurorobotics—a thriving community and a promising pathway toward intelligent cognitive robots *Front. Neurorob.* **12** 42

[16] Luque N R, Garrido J A, Carrillo R R, Tolu S and Ros E 2011 Adaptive cerebellar spiking model embedded in the control loop: context switching and robustness against noise *Int. J. Neural Syst.* **21** 385–401

[17] Ojeda I B, Tolu S and Lund H H 2017 A scalable neuro-inspired robot controller integrating a machine learning algorithm and a spiking cerebellar-like network *Conf. on Biomimetic and Biohybrid Systems* (Berlin: Springer) pp 375–86

[18] Furber S 2016 Large-scale neuromorphic computing systems *J. Neural Eng.* **13** 051001

[19] Schöner G 2016 *Dynamic Thinking* (*A Primer on Dynamic Field Theory*) (Oxford: Oxford University Press)

[20] Tieck J C V, Donat H, Kaiser J, Peric I, Ulbrich S, Roennau A, Zöllner M and Dillmann R 2017 Towards grasping with spiking neural networks for anthropomorphic robot hands *Int. Conf. On Artificial Neural Networks* (Berlin: Springer) pp 43–51

[21] Bing Z, Meschede C, Röhrbein F, Huang K and Knoll A C 2018 A survey of robotics control based on learning-inspired spiking neural networks *Front. Neurorob.* **12** 35

[22] Corchado C, Antonietti A, Capolei M C, Casellato C and Tolu S 2019 Integration of paired spiking cerebellar models for voluntary movement adaptation in a closed-loop neuro-robotic experiment. A simulation study *2019 IEEE Int. Conf. On Cyborg and Bionic Systems* (IEEE)

[23] Tieck J C V, Steffen L, Kaiser J, Roennau A and Dillmann R 2018 Controlling a robot arm for target reaching without planning using spiking neurons *2018 IEEE 17th Int. Conf. On Cognitive Informatics & Cognitive Computing (ICCI*CC)* (IEEE) pp 111–6

[24] Srinivasa N and Cho Y 2012 Self-organizing spiking neural model for learning fault-tolerant spatio-motor transformations *IEEE Trans. Neural Netw. Learning Syst.* **23** 1526–38

[25] Wu Q, McGinnity T M, Maguire L, Belatreche A and Glackin B 2008 2d co-ordinate transformation based on a spike timing-dependent plasticity learning mechanism *Neural Netw.* **21** 1318–27

[26] Bouganis A and Shanahan M 2010 Training a spiking neural network to control a 4-DoF robotic arm based on spike timing-dependent plasticity *The 2010 Int. Joint Conf. On Neural Networks (IJCNN)* (IEEE) pp 1–8

[27] Wolpert D M and Kawato M 1998 Multiple paired forward and inverse models for motor control *Neural Netw.* **11** 1317–29

[28] Blakemore S-J, Wolpert D and Frith C 2000 Why can't you tickle yourself? *Neuroreport* **11** R11–6

[29] Maravita A and Iriki A 2004 Tools for the body (schema) *Trends Cognit. Sci.* **8** 79–86

[30] Abadía I, Naveros F, Garrido J A, Ros E and Luque N R 2019 On robot compliance: a cerebellar control approach *IEEE Trans. Cybern.* 1–14

[31] Vannucci L, Falotico E, Tolu S, Cacucciolo V, Dario P, Lund H H and Laschi C 2017 A comprehensive gaze stabilization controller based on cerebellar internal models *Bioinspir. Biomim.* **12** 065001

[32] Chaumette F and Hutchinson S 2006 Visual servo control. I. Basic approaches *IEEE Robot. Autom. Mag.* **13** 82–90

[33] Kalaska J F 2009 From intention to action: motor cortex and the control of reaching movements *Progress in Motor Control* (Berlin: Springer) pp 139–78

[34] Amari S *et al* 2003 *The Handbook of Brain Theory and Neural Networks* (Cambridge, MA: MIT Press)

[35] Hodgkin A L and Huxley A F 1952 A quantitative description of membrane current and its application to conduction and excitation in nerve *J. Physiol.* **117** 500–44

[36] Abbott L F 1999 Lapicque's introduction of the integrate-and-fire model neuron (1907) *Brain Res. Bull.* **50** 303–4

[37] Izhikevich E M 2003 Simple model of spiking neurons *IEEE Trans. Neural Netw.* **14** 1569–72

[38] Hebb D 2002 *The Organization of Behavior 1949* vol 2 (New York: Wiely) p 8

[39] Oja E 1982 Simplified neuron model as a principal component analyzer *J. Math. Biol.* **15** 267–73

[40] Buonomano D and Carvalho T 2009 Spike-timing-dependent plasticity (STDP) *Encyclopedia of Neuroscience* ed L R Squire (New York: Academic) pp 265–8

[41] Caporale N and Dan Y 2008 Spike timing-dependent plasticity: a Hebbian learning rule *Annu. Rev. Neurosci.* **31** 25–46

[42] Gilson M, Fukai T and Burkitt A N 2012 Spectral analysis of input spike trains by spike-timing-dependent plasticity *PLoS Comput. Biol.* **8** e1002584

[43] Cutsuridis V, Cobb S and Graham B P 2008 A Ca^{2+} dynamics model of the STDP symmetry-to-asymmetry transition in the Ca1 pyramidal cell of the hippocampus *Int. Conf. On Artificial Neural Networks* (Berlin: Springer) pp 627–35

[44] Woodin M A, Ganguly K and Poo M-m 2003 Coincident pre- and postsynaptic activity modifies GABAergic synapses by postsynaptic changes in Cl-transporter activity *Neuron* **39** 807–20

[45] Buchanan K and Mellor J 2010 The activity requirements for spike timing-dependent plasticity in the hippocampus *Front. Synaptic Neurosci.* **2** 11

[46] Ruan H, Saur T and Yao W-D 2014 Dopamine-enabled anti-Hebbian timing-dependent plasticity in prefrontal circuitry *Front. Neural Circ.* **8** 38

[47] Zhang J-C, Lau P-M and Bi G-Q 2009 Gain in sensitivity and loss in temporal contrast of STDP by dopaminergic modulation at hippocampal synapses *Proc. Natl Acad. Sci.* **106** 13028–33

[48] Brzosko Z, Schultz W and Paulsen O 2015 Retroactive modulation of spike timing-dependent plasticity by dopamine *Elife* **4** e09685

[49] Hao Y, Huang X, Dong M and Xu B 2020 A biologically plausible supervised learning method for spiking neural networks using the symmetric STDP rule *Neural Netw.* **121** 387–95

[50] Izhikevich E M and Moehlis J 2008 Dynamical systems in neuroscience: The geometry of excitability and bursting *SIAM Rev.* **50** 397

[51] Izhikevich E M 2004 Which model to use for cortical spiking neurons? *IEEE Trans. Neural Netw.* **15** 1063–70

[52] Sridharan D and Knudsen E I 2015 Selective disinhibition: a unified neural mechanism for predictive and post hoc attentional selection *Vis. Res.* **116** 194–209

[53] Fidjeland A K, Roesch E B, Shanahan M P and Luk W 2009 Nemo: a platform for neural modelling of spiking neurons using GPUs *2009 20th IEEE Int. Conf. On Application-specific Systems, Architectures and Processors* (IEEE) pp 137–44

[54] Gamez D, Fidjeland A K and Lazdins E 2012 iSpike: a spiking neural interface for the iCub robot *Bioinspir. Biomim.* **7** 025008

[55] Hu M-K 1962 Visual pattern recognition by moment invariants *IRE Trans. Inf. Theory* **8** 179–87

[56] Sturm P 2014 *Pinhole Camera Model* (Berlin: Springer) pp 610–3

[57] Neymotin S A, Chadderdon G L, Kerr C C, Francis J T and Lytton W W 2013 Reinforcement learning of two-joint virtual arm reaching in a computer model of sensorimotor cortex *Neural Comput.* **25** 3263–93

[58] Navarro-Alarcon D and Liu Y-H 2018 Fourier-based shape servoing: a new feedback method to actively deform soft objects into desired 2D image shapes *IEEE Trans. Robot.* **34** 272–9