Learning target reaching motions with a robotic arm using dopamine modulated STDP

J. Camilo Vasquez Tieck[1], Pascal Becker[1], Daniel Reichard[1], Arne Roennau[1], Rüdiger Dillmann[1,2]

***Abstract*— The main purpose of the human arm is to reach a target and perform a manipulation task. Human babies learn to move their arms by imitating and doing motor babbling through trial and error. This learning is believed to result from changes in synaptic efficacy triggered by complex mechanisms involving neuromodulators in which dopamine plays a key role. After learning, humans are able to reuse and adapt the motions without performing complex calculations. In contrast, classical robotics achieve target reaching by mathematically computing each time the inverse kinematics of the joint angles leading to a particular target, then validating the configuration and generating a trajectory. This process is computational intensive and becomes more complex with the amount of degrees of freedom. In this work, we propose a spiking neural network architecture to learn target reaching motions with a robotic arm using reinforcement learning, which is closely related to the way babies learn. To make our approach scalable, we subdivide the kinematics structure of the robot and create one sub-network per joint. We generate training data offline by generating random reaching motions with an inverse kinematics calculation outside of the network. After learning, the inverse kinematics is no longer required, and the model is implicitly learned in the weights of the network. Mimicking the learning mechanisms of the brain, the synaptic plasticity learning rule is STDP modulated by dopamine, representing a reward. The approach is evaluated with a simulated Universal Robot UR5 with six degrees of freedom. The network successfully learns to reach multiple targets and by changing the reward function on-the-fly it is able to learn different control functions. With a standard computer our network was able to control a robotic kinematics chain up to 13 degrees of freedom in real time. We believe that model free motion controllers inspired on the human brain mechanisms can improve the way robots are programmed by making the process more adaptive and flexible.**

I. INTRODUCTION

The main purpose of the human arm is to reach a target and perform a manipulation task. In comparison to robotic applications humans do not compute inverse kinematics to perform arm movements directly. The arm is moved towards the goal until it reaches the object based on former experiences and also on sensory input for example from the eyes or finger tips. Human babies learn to move their arms by imitating and doing motor babbling through trial and error. This learning is believed to result from changes in synaptic efficacy triggered by complex mechanisms, one of them being induced by dopamine. After learning, humans are able to reuse and adapt the motions without performing complex calculations. The human being is able to learn almost any physical change of the arm and the resulting movement within few iterations and adapt the future movements to that change.

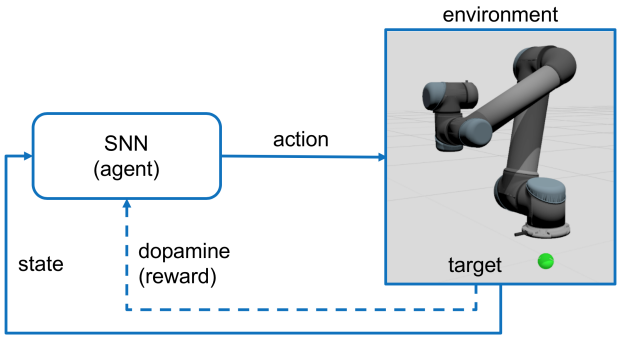


Fig. 1: General view of the architecture. A SNN learns target reaching motions to control a robotic arm for using reinforcement learning with dopamine.

The development of robotic applications advances every day with a multitude of robotic arm types with different kinematics. Their main goal is reaching a target for executing a task with a manipulator. In contrast to humans, classical robotics achieve target reaching by mathematically computing each time the inverse kinematics of the joint angles leading to a particular target, then validating the configuration and generating a trajectory. This calculation requires a kinematic model of the robot and it is computational intensive becomes more complex with the amount of degrees of freedom.

The principle of reinforcement learning (RL) [1] is simple, almost natural and can be observed even in the behavior of animals or humans during their babyhood [2]. The last years have witnessed a huge growth in applications for RL in different domains. From human like behavior to control Atari games [3], to applied robotics tasks. In robotics it has been used to learn robotic manipulation [4], to learn and adapt motions [5], to train dynamic models in closed-loop for manipulation [6], to name some examples.

Recent developments in the field of spiking neural networks (SNN) [7]–[9] have led to insights of brain mechanisms such as complex learning rules [10] and learning modulation. Dopamine has been identified as being responsible for playing an important role in modulating plasticity, and its behavior has been modelled with SNN [11]. SNN focus on the biological characteristics of neurons, and model closer the way real neurons work. Exploring the capabilities of SNN, enables research on the learning mechanisms and information representation in the brain. Various approaches have been proposed to use SNN in robotics such as model

[1] FZI Research Center for Information Technology, 76131 Karlsruhe, Germany. **tieck**, pbecker, jkaiser, dreichar, roennau@fzi.de

[2] Karlsruhe Institute of Technology (KIT), Germany. ruediger.dillmann@kit.edu

free control of an arm using motor primitives [12], [13] or using RL to learn continuous muscle control [14], or using RL to learn ball balancing [15], or using RL with dopamine to learn vehicle control [16].

Few researchers have addressed the problem of combining SNN with RL and robotics. Our approach is inspired on the brain mechanisms for how babies learn to move their arms by doing motor babbling and imitating what they see.

In this work, we propose a spiking neural network architecture to learn target reaching motions with a robotic arm using reinforcement learning. The main components of our approach are presented in Fig. 1. To make our approach scalable, we sub-divide the kinematics structure of the robot and create one sub-network per joint. We generate training data offline by generating random reaching motions with an inverse kinematics calculation outside of the network. After learning, the inverse kinematics is no longer required, and the model is implicitly learned in the weights of the network. To represent the reward we use a dopamine mechanism to modulate STDP plasticity, mimicking the learning mechanisms of the brain. The network is able to learn and adapt to different tasks with a change of the reward function and the behavior of the robot is completely changed without changing the network. We benchmark our approach to test the real time control capabilities with relation to the training data and the amount of degrees of freedom.

II. APPROACH

In this section we describe our bio-inspired approach for learning target reaching motions with a robotic arm using reinforcement learning with spiking neural networks (SNN). Our approach has four main components — the sensory populations, the action populations, the reward populations and the state representation. A detailed view of the SNN architecture to control one joint with all components is presented in Fig. 2. To represent the reward we use dopamine to modulate STDP plasticity resembling brain mechanisms. We divide the kinematics model of the robot and create one sub-network per joint. The SNN is able to learn and relearn different behaviors. The inverse kinematics to define the joint angle targets is calculated outside the SNN.

Sensory input for the target position is represented with two populations one for smaller and greater angle values (sensory populations). The action is represented with two populations to determine the next movement direction, either up or down (action population). The decoding for movement calculation sums the spikes. Inhibitory connections between the action populations improves the outcome of the learning as a winner-takes-it-all circuit. Two populations represent the reward using dopamine, the incoming synapses of the up and down population (reward populations). This populations are used to learn and unlearn different behaviors. To represent the state and mimic how the visual input and the proprioception are processed we use a SNN subtract block [17].

The proposed network architecture is able to adapt the behavior of the robot on-the-fly. That is possible as the reward function of the reinforcement learning defines which actions lead to a reward and which not and the function can be modified easily during runtime. As long as dopamine is distributed towards the synapses they are able to adapt their weights and learn the correlation between sensory and action populations.

A. Sensory and action populations

Two sensory populations represent the current state of the joint with respect to the target position, one for smaller and one for greater angle values. If the current joint angle is greater than the given target angle the greater population is stimulated, or if the current joint angle is smaller the smaller population is stimulated.

The action for one joint is represented with two populations to determine the next movement, either up or down. Both action population are connected all-to-all to both sensory populations with dopamine STDP synapses. To decode the spike activity and generate motor commands, the amount of spikes S of each population $[up, down]$ is counted during one simulation step. A delta $\Delta\theta$ per joint is calculated and it is added to the current joint angle to obtain the next position command θ_{t+1} as:

$$\Delta\theta_{t+1} = S_{up}(t) - S_{down}(t) \tag{1}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_{t+1} \tag{2}$$

This representation is a simplified model of the agonist antagonist muscle synergies. The output of both populations can also be interpreted as the change of length of two biological muscles around one joint, if the flexor increases its length, the extensor gets shorter and vice versa. To facilitate this relation both populations are connected with inhibitory synapses with each other as in a winner-takes-all circuit.

B. Reward populations and dopamine synapses

The sensory and action populations are connected with dopamine-based STDP synapses, and they do not learn until there is dopamine distributed to the synapse. Two populations represent the reward using dopamine. This mechanism is illustrated in Fig. 3. The reward populations are activated if a reward is received. Their activation represents the amount of dopamine in the synapse and it modulates the STDP plasticity triggering learning [11]. The connections between the reward populations and the dopamine synapses are static, so the weights are fixed.

The dopamine-based modulation plays an important role in this approach as it is the link between SNN and reinforcement learning. We used the NEST simulator [18] to model this mechanism. The *stdp_dopamine_synapse* model is used for the connections between the sensory and the action populations [11]. An important detail here is the fact that a neuronal population is directly connected to a synapse instead of another population using the *volume_transmitter* interface.

C. Reinforcement learning modelling

Reinforcement learning [1] can be modelled as a Markov decision process (MDP)

$$MDP = (S, A, P(s,s'), R(s,s'), \gamma) \tag{3}$$

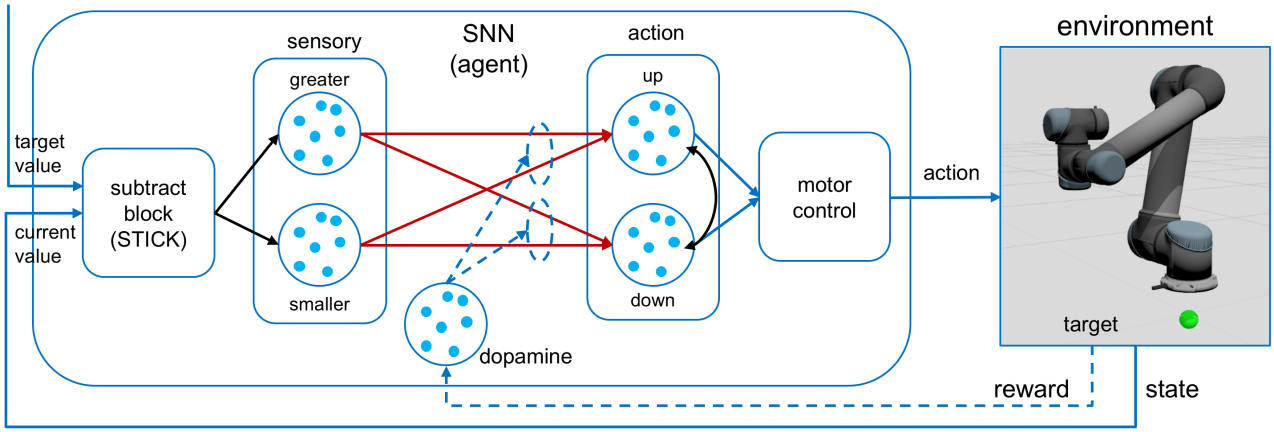


Fig. 2: Detailed view of the architecture of the spiking neural network architecture to control one joint.

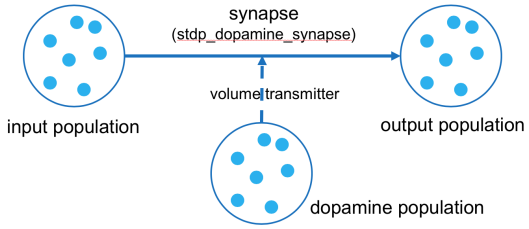


Fig. 3: Dopamine modulated STDP. Implemented using the *stdp_dopamine_synapse* model from NEST [18].

with a finite set of states S and actions A, a policy P, a discount factor γ and a reward function R. There are two states in S for each joint: greater and smaller than the target angle. They are represented by the two sensory populations in the SNN. As the agent can be only in one state at a time either one or the other population is stimulated. The state is set with incoming spikes and the selected population spikes to stimulate the corresponding action. There are only two possible actions in A for each joint: up and down. The amount of spikes in both action populations is compared to determine the next action and the delta to move the joint. The policy P defines which action is chosen in each state. The policy is encoded in the synaptic weights between the sensory and action populations. The weight is proportional to the probability that the postsynaptic action is executed. The discount factor γ is a parameter between 0 and 1 that determines the influence of future rewards. In combination with SNN and dopamine-modulated STDP the discount factor is encoded as an eligibility trace. The differences in spike timing increases or decreases the eligibility trace of the given synapse which leads to a weight change if dopamine is received.

The reward R functions plays an important role in this approach as it allows us to change the behavior of the robot on-the-fly. We define three different reward functions, one to move towards the target, one to move away from the target, and we added another function to disable learning. The reward is calculated by the change in the difference between the current and target angles from the current and the last simulation step. To move towards the target the network is rewarded when the difference gets smaller. In this case, the target is defined as reached when the distance is smaller than 5.0 degrees over all joints. To move away from the target the network is rewarded when the difference gets larger. In this case, the target is defined as reached when the distance is greater than 120.0 degrees over all joints. To disable learning the network receives no reward, and thus stops the learning. This kind of functionality is useful to fix and maintain the current weights after a learning period.

D. State representation

To represent the state, and mimic how the visual input and the proprioception are processed in the brain. The Spike Time Interval Computational Kernel (STICK) [17] presents several SNN architectures to perform general purpose computations. Internally STICK uses integrate-and-fire neurons without the decay of the membrane potential and with exponential post-synaptic current dynamics. One of the networks proposed in STICK is a subtract block and it is shown in Fig. 4.

The subtract block takes two inputs, which in our case are current and target angle values, and returns an output after a time of 110ms as the subtract network encodes the result using precise time encoding. The positive output neuron spikes when the difference is greater than zero and the negative output neuron spikes if the difference is smaller than zero. The output neurons of the subtract block are connected to the sensory populations of the joint network with fixed weights.

III. IMPLEMENTATION

In this work we use a Universal Robot UR5 with six degrees of freedom. The software architecture is presented in Fig. 5 and it uses three software frameworks: the Neural Simulation Tool (NEST) [18] for the simulation of the spiking neural networks, Gazebo to visualize the target and the robotic arm, and the *Robot Operating System (ROS)*[*] as a middleware to connect NEST with Gazebo and control the robot. Each robotic joint is controlled by its own sub network. The advantage of this factorized approach is, that it is easy to adapt the proposed model with different

[*] http://www.ros.org/

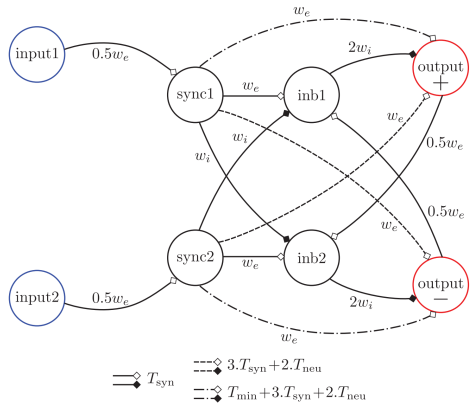


Fig. 4: Subtract block implemented with STICK. A SNN that calculates the difference between two values using precise time coding. (adapted from [17]).

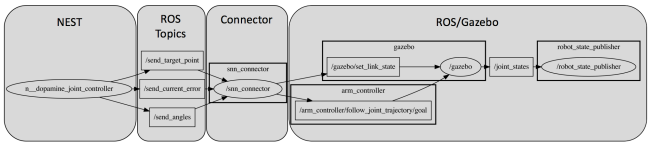


Fig. 5: Pipeline. Software stack and information flow.

robotic hardware as the amount of joints is not fixed. The inverse kinematics to define the reference joint angle targets is calculated outside the SNN using ROS stack with Trac-IK [19]. As the robot uses a standard robot controller it is easily replaceable with any other robot type, and after evaluating the experiments in simulation the real robot can be used without big changes in the software. The network blocks for STICK proposed in [**?**] allow us to perform exact mathematical computations with SNN. A python implementation with NEST called pyStick [20] was developed at the FZI †.

The SNN is implemented in NEST and the dopamine synapse model is based on the work proposed by Izhikevic [11], implemented by Potjans [21].The parameters for the *integrate-and-fire* neuron model: E_L -70.0, C_m 250.0, τ_m 10.0, t_{ref} 2.0, V_{th} -55.0, V_{reset} -70.0, τ_{syn} 2.0, I_e 0.0. The parametres for the *stdp_dopamine_synapse* model: W_{min} 10.0, W_{max} 1000.0, A_{plus} 1.2, A_{minus} 1.1, τ_{plus} 20.0, τ_c 150.0, τ_n 10.0, b 0.0, c 0.0, n 0.0. All populations use the same parameters with the exception of: the sensory populations use Cm = 100.0 and m = 10.0; the action and reward populations use Cm = 50.0 and m = 50.0.

The sequence diagram in Fig. 6 shows the process of initialization, setting a target and simulating the joint movements. The DopamineJointController is the main actor and orchestrates all joints needed to control the robot. After the initialization when a target was set, the inverse kinematics for the robot and the given pose in spatial coordinates is calculated with Trac-IK. This process returns the target joint angles to all joints. In each simulation step all networks are simulated at once during the *nest.simulate()* function call. Afterwards, networks for each joint are processed in sequence and their spikes are read out to calculate the new joint angle. After all joints have been processed the new joint values are sent to the ROS stack to visualize them in Gazebo.

†FZI Research Center for Information Technology

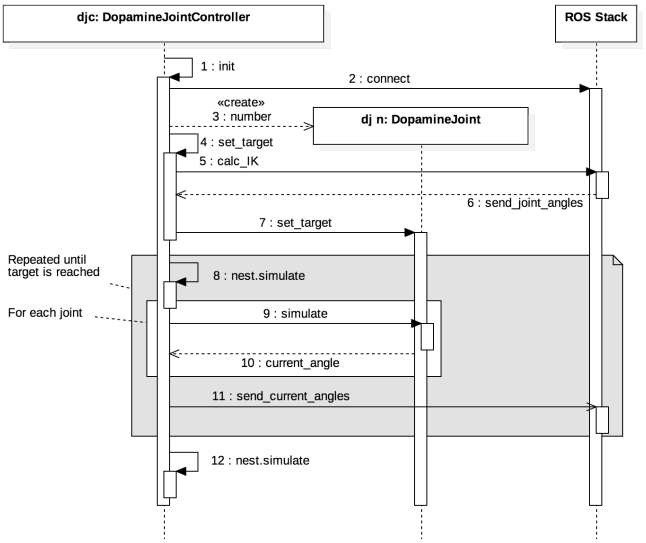


Fig. 6: Sequence diagram. Detailed process of initialization, setting a target and simulating the joint movements.

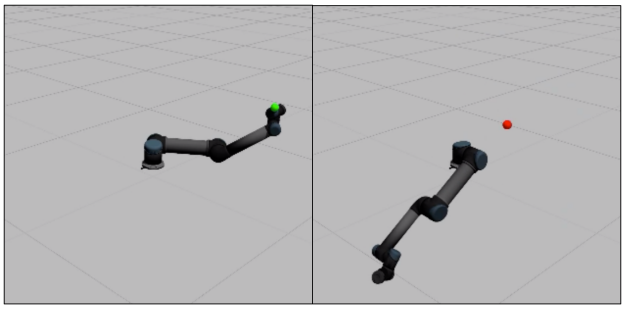


Fig. 7: Experiment setup with a six DoF robotic arm. Moving towards (*left*) and moving away (*right*) from the target.

IV. RESULTS

Our experimental set up is inspired on the way babies learn to move their arms by imitating and doing motor babbling. A robotic arm in simulation learns how to move towards or away from a target (see Fig. 7). In this section we present the experiment setup, the evaluation, and the definitions for the metrics of when a target is reached and how the distance and the time are measured. The benchmarks directly depend on the used hardware, so we present also the specifications of the computers used for the experiments.

A. General experiment setup

All the experiments were carried out in simulation as illustrated in Fig. 7. We used a model of a Universal Robot UR5 with six degrees of freedom. A sphere represents the target the robot has to reach. The starting point for all joints was always 0.0 degrees and a fixed seed guaranteed always the same initial random weights for the dopamine synapses.

To obtain comparable evaluation results we generated two data sets — one for the learning and one for the testing phase. Both consist of 6D poses (x, y, z, roll, pitch, yaw) as target for the robot. All targets were drawn from a uniform distribution in the range of the robot $[-0.6, 0.6]$ to have high chances of getting a valid configuration. An inverse kinematics calculation with Trac-IK was done to check if the target is reachable with the end-effector of the robot. Only points with a valid inverse kinematics solution were selected.

We tested the approach on three different machines all running Ubuntu 14.04 64-bit. Configuration 1 was CPU Intel i7-4770 (Quad Core), RAM 8192 MB, GPU NVIDIA GTX TITAN. Configuration 2 was CPU Intel i3-2100 (Dual Core), RAM 4096 MB, GPU NVIDIA GTX 550 Ti. Configuration 3 was CPU Intel i7-5820K (Hexa Core), RAM 16384 MB, GPU NVIDIA GTX 970.

B. Metrics

A target is defined as reached for one time step if the mean distance to target of all joints of the robotic arm is within a given range. The mean distance to target d_{total} is calculated as

$$d_{total} = \frac{\sum_i^N d_i}{N} \tag{4}$$

with N representing the amount of joints. To prevent lucky solutions, the arm has to maintain the position bellow a given range for 100 time steps. In each time step either the arm is at the target or it is not. If it is, a target counter is increased. If the arm reached a target but then missed it for continuous 50 time steps the target counter is reset to 0. The purpose of this reset mechanism is to set a maximum time limit the robot can be off the target and to prevent reaching a target just by chance. If the arm is not able to reach the target within 1000 time steps (150 seconds) the target counts as not reached and the experiment continues with the next target.

The measured time is in simulation cycles, each cycle is 150 milliseconds. Each measurement starts with the setting of the target and stops when the target is reached. In general, the time for reaching a target is determined by the slowest moving joint in the kinematics structure.

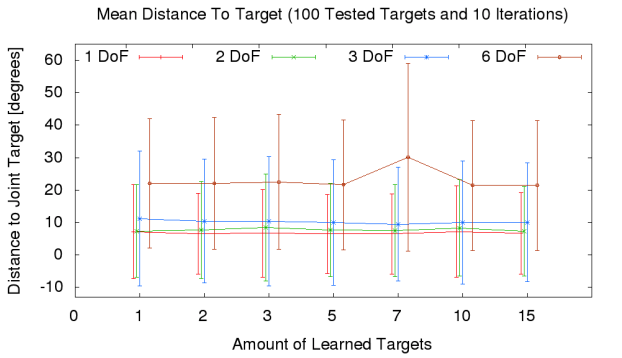
While testing one target, the difference between current and target angle values is measured and stored for error calculation after the target was reached. For error calculation the standard deviation σ is calculated as

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{5}$$

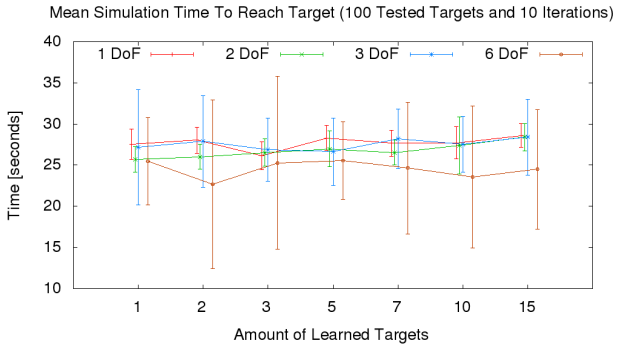
with N representing the amount of dopamine joints, x_i the current distance and $\bar{x}$ the mean of all values.

C. Evaluation

To benchmark the performance, we conducted several runs of the experiment changing the amount of learned targets as well as the amount of degrees of freedom. To prevent outliers or lucky shots every combination ran 10 times with 100 targets each and the mean value of all was plotted.



(a)



(b)

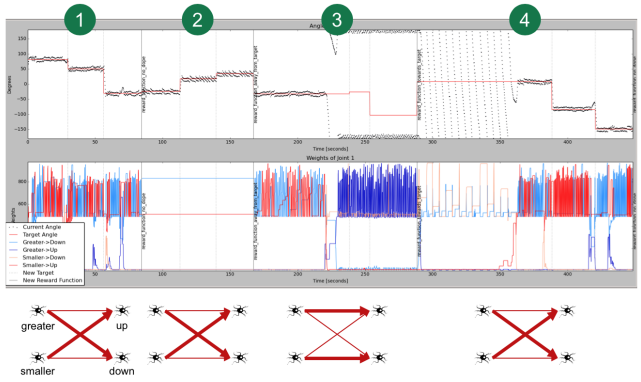
Fig. 8: Performance depending on the amount of initially learned targets and degrees of freedom. (a) Mean distance to target. (b) Mean time needed to reach the target.

In Fig. 8a we show the mean distance to target with the standard deviation for all joints depending on the amount of initially learned targets and degrees of freedom. We observe that the mean distance to target increases with the amount of degrees of freedom, that means that the complexity of the kinematic structure increases the error. We also observe that the mean distance to target and the standard deviation remain almost constant with the amount of learned targets, which is to be expected because each joint runs in parallel.

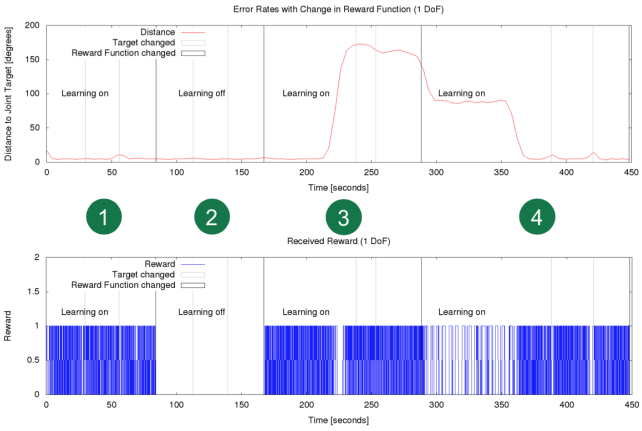
In Fig. 8b we show the mean time needed to reach the target depending on the amount of initially learned targets and degrees of freedom. We observe that the standard deviation increases with the amount of degrees of freedom, that means that the complexity of the kinematic structure makes it harder to find a proper configuration. We also observe, that the mean time has its minimum around three to five targets depending on the degrees of freedom.

The evaluation was done with a all six degrees of freedom to test the factorized kinematics approach. The plots in Fig. 9 show four phases of learning for one degree of freedom: (1) Learn movements towards targets, (2) Execute movements towards targets (without dopamine), (3) Learn movements away from targets, (4) Learn movements towards targets.

In the first phase, the network learns to move towards the target. During this phase the weights are adapted according to their eligibility trace as the reward function distributes



(a)



(b)

Fig. 9: Learning and unlearning to change the behavior during the execution of different target reaching tasks with different reward functions. (a) Current and target joint angle and the synaptic weights. (b) Distance to target of the joint angle and granted reward.

dopamine to the synapses every time the arm moves towards the target angle. The changes in the synaptic weights are illustrated with the red arrows in the bottom of Fig. 9b. In the second phase, the learned behavior is tested by keeping the weights constant. This is done by changing the reward function to the one that does not distribute dopamine to the synapses at all, so the learning is turned off. In the third phase, the reward function is changed again and the network is rewarded when it moves away from the target. In the fourth phase, the network learns again to move towards the target, to show that the network is able to relearn the behavior at any time.

One of the main features of our approach is to be able to change the behavior on-the-fly by unlearning the old and learning a new one using the three different reward functions. This changes of reward functions can take place at any time, but it takes time to adapt the weights, as shown in Fig. 9b. In Fig. 9a we show the error of the joint angle in relation to the given target angle while executing multiple target reaching tasks with the different reward functions.

Robots need real time control, so the network simulation has to be at least as fast as real time. NEST takes 150 milliseconds per simulation step (cycle), so we have 7 cycles

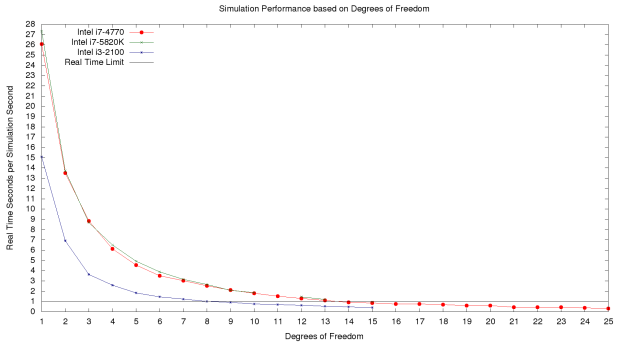


Fig. 10: Real time factor. Relation of the amount of degrees of freedom with respect to the real time factor evaluated in three different machines.

to simulate to reach real time. In the given setup the hardware was able to control up to 13 degrees of freedom as shown in Fig. 10. The current NEST implementation of the dopamine synapse is not able to use multi-threading, this could change in the future to obtain a greater speedup.

V. DISCUSSION

In this work we presented a spiking neural network architecture to learn and perform target reaching tasks with a robotic arm using dopamine modulated STDP. Our approach is scalable to control a n-DoF robotic arm, and is able to change on-the-fly the control behavior.

A reinforcement learning agent and a SNN were combined feeding the reward signal as dopamine into the network. The synaptic weights only changes when the neurotransmitter dopamine is received by the synapse otherwise no learning will take place. As the dopamine is distributed by the agents reward function we were able to change the reward function during runtime.

The network is able to relearn the target function during run-time without any external changes in the network. In our work we had two different behaviors: moving towards targets and moving away from targets. Both were learned and unlearned several times and the network was able to change its behavior every time.

The network size is linearly dependent on the amount of degrees of freedom of the robot kinematics. This factorized kinematics model where each joint is represented by a sub network deals easily with more dimensions. In the evaluation we were able to control up to 13 joints in real time with the proposed approach.

In case of dopamine modulation a volume transmitter is directly connected to the dopamine synapse to deliver the reward signal. This could be the reason why experiments showed that the *stdp_dopamine_synapse* synapse model in NEST is not compatible with multi-threading. A version with working multi-threading dopamine synapses should speed up the simulation.

We currently work on running the SNN on SpiNNaker [22] neuromorphic hardware to improve the speed. With SpiNNaker it would be possible to train the network once

and embed the trained spiking neural network in the robot as a controller.

The movement in each simulation step is calculated by the difference of spikes in the action populations, big differences lead to a fast movement of the arm. To get smoother movements we could change the architecture so that the control is adaptive and the arm moves fast towards the target when it is further away and reduces the speed the closer it gets. Especially, when it is close to the target and adding a cerebellum model [23] for finer controlling and as a movement filter.

The inverse kinematics calculations in this work were done by a numerical approach called Trac-IK [19], nevertheless research [24] showed that it is also possible to use a SNN to learn this complex mapping of poses to joint angles. This neural approach could be added as a separate component. Adding this network makes it possible to compare the whole approach with the behavior in humans as they do not have a numerical inverse kinematic for joint angle calculation.

The proposed network architecture of this work could be implemented in the *Neurorobotics Platform (NRP)*[‡] [25] of HBP and would enable more researchers to perform simple target reaching experiments. Although, as of now, the method of delivering spikes to a synapse directly is not the standard approach and it is currently not possible to use the dopamine synapse model on NRP.

Target reaching is only one small problem of robotic applications. Tasks like obstacle avoidance or more sophisticated problems can not be solved by the proposed approach. But still, dopamine modulation in combination with reinforcement learning could be used to learn and change the behavior of the robot during runtime.

ACKNOWLEDGMENT

This research was supported by the HBP Neurorobotics Platform and received funding from the European Unions Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreement No. 785907 (Human Brain Project SGA2).

REFERENCES

[1] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[2] A. Lotem and D. W. Winkler, "Can reinforcement learning explain variation in early infant crying," *Behavioral and Brain Sciences*, vol. 27, no. 4, pp. 468–468, 2004.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[4] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[5] J. Kober, E. Oztop, and J. Peters, "Reinforcement learning to adjust robot movements to new situations," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[6] T. G. Thuruthel, E. Falotico, F. Renda, and C. Laschi, "Model-based reinforcement learning for closed-loop dynamic control of soft robotic manipulators," *IEEE Transactions on Robotics*, vol. 35, no. 1, pp. 124–134, 2019.

[7] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, 1997.

[8] J. Vreeken, "Spiking neural networks, an introduction," 2003.

[9] A. Grüning and S. M. Bohte, "Spiking Neural Networks: Principles and Challenges," in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning – ESANN*, 2014.

[10] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic Plasticity Dynamics for Deep Continuous Local Learning," *arXiv preprint arXiv:1811.10766*, 2018.

[11] E. M. Izhikevich, "Solving the distal reward problem through linkage of stdp and dopamine signaling," *Cerebral cortex*, vol. 17, no. 10, pp. 2443–2452, 2007.

[12] J. C. V. Tieck, L. Steffen, J. Kaiser, D. Reichard, A. Roennau, and R. Dillmann, "Controlling a robot arm for target reaching without planning using spiking neurons," in *2018 IEEE 17th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC)*, 2018.

[13] ——, "Combining Motor Primitives for Perception Driven Target Reaching With Spiking Neurons," *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)*, vol. 13, no. 1, p. 12, 2019.

[14] J. C. V. Tieck, M. V. Pogančić, J. Kaiser, A. Roennau, M.-O. Gewaltig, and R. Dillmann, "Learning Continuous Muscle Control for a Multi-joint Arm by Extending Proximal Policy Optimization with a Liquid State Machine," in *International Conference on Artificial Neural Networks ICANN 2018*. Springer, 2018, pp. 211–221.

[15] J. Kaiser, M. Hoff, A. Konle, J. C. V. Tieck, D. Kappel, D. Reichard, A. Subramoney, R. Legenstein, A. Roennau, W. Maass, and R. Dillmann, "Embodied Synaptic Plasticity with Online Reinforcement learning," *Frontiers in Neurorobotics*, 2018 (submitted).

[16] Z. Bing, C. Meschede, K. Huang, G. Chen, F. Rohrbein, M. Akl, and A. Knoll, "End to end learning of spiking neural network based on r-stdp for a lane keeping vehicle," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

[17] X. Lagorce and R. Benosman, "Stick: spike time interval computational kernel, a framework for general purpose computation using neurons, precise timing, delays, and synchrony," *Neural computation*, vol. 27, no. 11, pp. 2261–2317, 2015.

[18] M. Diesmann and M.-O. Gewaltig, "NEST: An environment for neural systems simulations," *Forschung und wisschenschaftliches Rechnen, Beitrage zum Heinz-Billing-Preis*, 2001.

[19] P. Beeson and B. Ames, "Trac-ik: An open-source library for improved solving of generic inverse kinematics," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2015, pp. 928–935.

[20] I. Peric, F. Schneider, C. H. Price, S. Ulbrich, A. Roennau, M. Zoellner, and R. Dillmann, "Exact spike timing computational model of convolutional associative memories," in *2017 IEEE 16th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*. IEEE, 2017, pp. 182–190.

[21] W. Potjans, A. Morrison, and M. Diesmann, "Enabling functional neural circuit simulations with distributed computing of neuromodulated plasticity," *Frontiers in computational neuroscience*, vol. 4, p. 141, 2010.

[22] S. B. Furber, S. Temple, and A. Brown, "High-Performance Computing for Systems of Spiking Neurons," in *AISB'06 Workshop. GC5: Archit. Brain Mind*, 2006.

[23] I. B. Ojeda, S. Tolu, M. Pacheco, D. J. Christensen, and H. H. Lund, "A combination of machine learning and cerebellar-like neural networks for the motor control and motor learning of the fable modular robot," *Journal of Robotics Networks and Artificial Life*, vol. 4, no. 1, pp. 62–66, 2017.

[24] E. Rueckert, D. Kappel, D. Tanneberg, D. Pecevski, and J. Peters, "Recurrent spiking networks solve planning tasks," *Scientific reports*, vol. 6, p. 21142, 2016.

[25] E. Falotico, L. Vannucci, A. Ambrosano, U. Albanese, S. Ulbrich, J. C. V. Tieck *et al.*, "Connecting artificial brains to robots in a comprehensive simulation framework: The neurorobotics platform," *Frontiers in Neurorobotics*, 2017.

[‡] http://www.neurorobotics.net/