# Virtual Neurorobotics in the Human Brain Project

Max Becker, Olivier Boder and Qianlin Wu
FZI Forschungszentrum Informatik
Karlsruhe, Germany

***Abstract*—Within the virtual neurorobotics Praktikum, the goal is to grab and throw a cylinder with a robot arm as far as possible. In order to achieve this, several solutions are discussed. One approach is to use hard-coded configurations involving inverse kinematics to grasp the cylinder at random positions and throw it with a predefined movement. Another more sophisticated approach is to learn throwing a cylinder with spiking neural networks. Therefore, the synaptic weights are learned with evolutionary algorithms. Given the experiment environment, the results are rather reasonable and show that the task can be achieved.**

***Index Terms*—Spiking Neural Networks, Evolutionary Algorithms, Neurorobotics Platform, Human Brain Project**

## I. Introduction

In the scope of the *Praktikum* within the Human Brain Project, we work with the Neurorobotics Platform (NRP) which provides a simulation environment for robotics. More specifically, our experiment setup consists of a table on which a robot arm is placed. The robot arm has six joints and a hand with five fingers. In front of the robot arm, a cylinder is placed on the table. The goal of the Praktikum is to move the cylinder as far as possible using the robot arm. This can be performed by just hitting the cylinder away or by grabbing the cylinder and throwing it as far as possible. The performance is measured by the distance of the cylinder from the table. The setup is depicted in Figure 1.

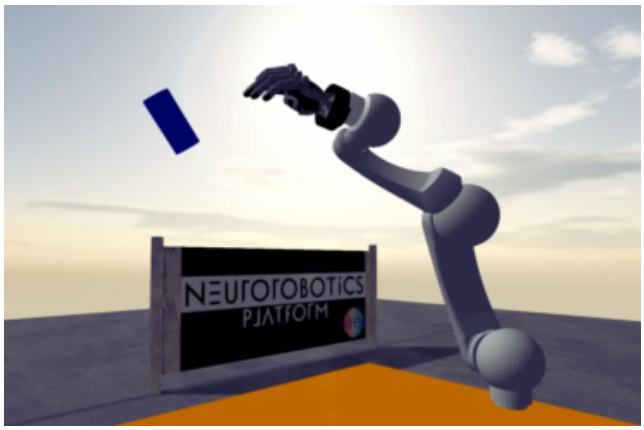


Fig. 1. The goal of the challenge is to throw the cylinder as far as possible.

In the following, several strategies to fulfill this task are presented. To begin with, fundamentals of evolutionary algorithms and spiking neural network are explained in section II respectively section III. Subsequently, a hard-coded approach in section IV followed by a more sophisticated learning approach with evolutionary algorithms in section V are presented to solve the challenge. Then, section VI depicts some obstacles that arose during the experiments and the Praktikum. Finally, the results are presented in section VII and a conclusion is drawn in section VIII

## II. Evolutionary Algorithms

Evolutionary algorithms are inspired by nature and evolution. They try to approximate a solution following the principle of survival of the fittest. In fact, an evolutionary algorithm models a population consisting of several individuals. Each individual represents a solution to a task, such as an optimization problem, and is evaluated by a fitness function. One populations is called a generation. The elite of each generation, i.e. the individuals performing best according to their fitness, is then selected to evolve the next generation. This allows the population to evolve towards better solutions. The evolution of a population corresponds to the mutation of individuals. The evolution process is depicted in Figure 2.

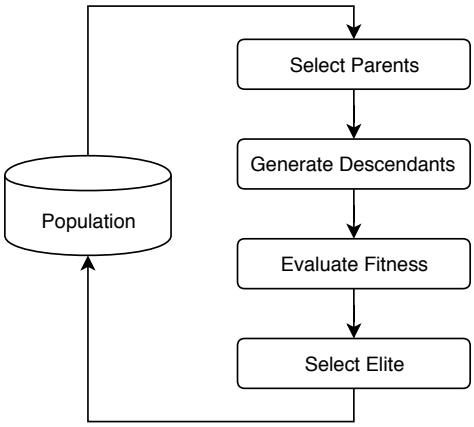


Fig. 2. An illustration of an evolutionary algorithm learning. [1]

To mutate individuals, the crossover strategy can be applied. Crossover is used to generate new offspring by combining the genetic information of two parents. One way to combine their genetic information is single-point crossover. The genetic sequences of both parents are split at a randomly chosen point. The second part, i.e. the tail, is then swapped between the parents. Hence, each new individual has its first parents genetic information until the chosen point, followed by the genetic sequence of the other parent. To apply more variety into the

recombination of the genetic information k-point crossover can be applied. Instead of only one point, more such random points are picked in the sequence, at which alternately information is swapped between the parents. [1]

As individuals are evolving over generations, they may improve their fitness and thus optimize a solution for the task. Their genetic information can therefore be parameters to solve a problem. Overall, there is no specific or correct parameter assumption needed for initialization, as they will be approximated by evolving over generations.

## III. Spiking Neural Networks

Spiking neural networks try to mimic natural neural networks more closely than other artificial neural networks. The neurons in spiking neural networks are connected via synapses which are weighted and organised in a directed graph. Every neuron in the network has a membrane potential. If this potential exceeds a certain threshold, the neuron fires, meaning it sends a spike to all its succeeding neurons in the network. The potential of the spiking neuron is then reset and, depending on the implementation, the neuron may enter a short refractory period in which it cannot fire again. If a neuron receives a spike from a predecessor, its membrane potential rises or falls depending on the synapse being inhibitory or excitatory. In the periods between spikes, the neurons leek some of their potential [2].

To learn with spiking neural networks, the weights of the synapses have to be changed. These weights determine how much influence a spike has on the potential of a succeeding neuron. In contrast to other artificial neural networks, the neurons don't have to be organized in strict layers and not all neurons in a layer need to be computed at the same time. The information is less encoded in the neurons values and more in the timing of the spikes [3]. However, it is harder to learn with spiking neural networks, because the spikes aren't differentiable, which means the weights cannot be learned using error backpropagation. One possibility to learn the weights are evolutionary algorithms.

Spiking neural networks are a good option for robot controlling tasks as they try to simulate biological neural networks as similarly as possible and offer advantages in speed, energy efficiency and computation capabilities [4].

## IV. Hard-coded Approach

In the hard-coded approach various states are defined each corresponding to a pose of the robot arm. A state machine cycles through these states and sends control messages to two scripts controlling the robot. The positions are initially determined empirically and later generalised with a set of equations to enable the robot to grab the cylinder from different positions on the table.

### A. State Machine

Figure 3 shows the 5 states used to control the robots movement: *Approach*, *Grasp*, *Prepare*, *Throw* and *Release*. The first two states let the robot approach and grasp the cylinder. In *Prepare* the robot reaches back to prepare for its throw. In the last two states the robot executes the actual throwing motion and finally releases the cylinder. Each time a new state is reached, the state machine sends a message via ROS to the respective control script: *Hand Control* or *Arm Control*. These control scripts then choose a configuration according to the current state and adjust the robots pose.

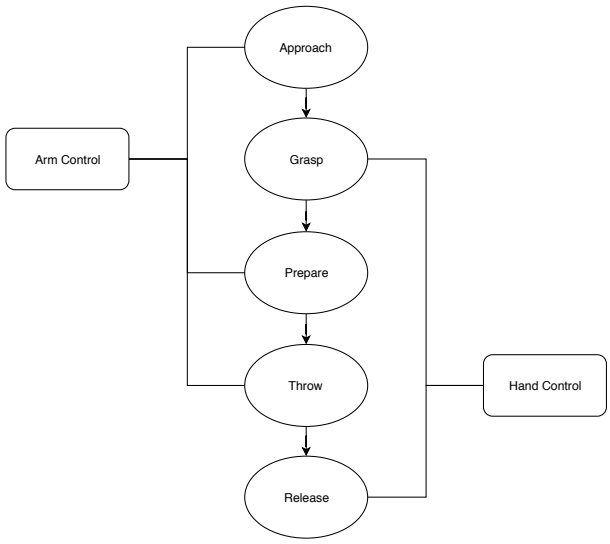


Fig. 3. The state machine controls the behavior of the robot.

### B. Inverse Kinematics

To grab the cylinder from different positions on the table, the robots inverse kinematics are used to determine the joint angles. When a new cylinder is spawned in the simulation, its position gets published to a ROS topic. If the cylinder position is $(C_x, C_y, C_z)$ and the robots position is $(R_x, R_y, R_z)$, the angle $\alpha$ for the first joint (see Figure 4) can be calculated as follows:

$$M_1 = \alpha = arctan((C_y - R_y)/(C_x - R_x)) \quad (1)$$

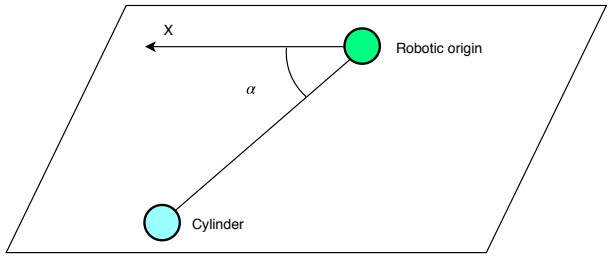


Fig. 4. The platform with the robot arm and the cylinder seen from above.

Using the empirically determined configuration for the initial hard-coded grasp, the length of the robot arm segments *Arm_2* and *Arm_4* can be calculated (Equation 2). For reference, see Figure 5.

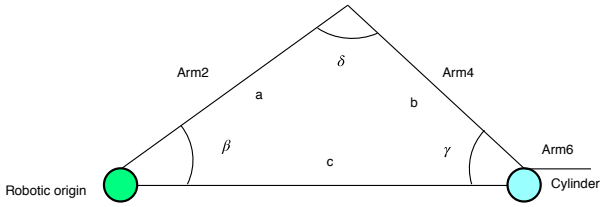


Fig. 5. The platform with the robot arm and the cylinder seen from the side.

$$\begin{aligned} \delta &= 2 * \pi - \beta - \gamma \\ c &= \sqrt{(R_x - C_x)^2 + (R_y - C_y)^2} \\ \textit{Arm\_2} &= c * sin(\gamma)/sin(\delta) \\ \textit{Arm\_4} &= c * sin(\beta)/sin(\delta) \end{aligned} \tag{2}$$

With the lengths of the arm segments now known, the remaining angles can be computed using Equation 3. The angles for the third and fifth joint are 0.

$$\begin{aligned} M_2 = \beta &= arccos((a^2 + c^2 - b^2)/2 * a * c) \\ M_4 = \pi - \delta &= \pi - arccos((a^2 + b^2 - c^2)/2 * a * b) \\ M_6 = \gamma &= arccos((b^2 + c^2 - a^2)/2 * b * c) \\ \textbf{with } c &= \sqrt{(R_x - C_x)^2 + (R_y - C_y)^2} \end{aligned} \tag{3}$$

## V. Learned Approaches

The next approach tries to learn some of the movements instead of hard-coding them. An evolutionary approach is used to learn weights for a spiking neural network which controls the robots movements.

### A. Evolutionary Approach

In this approach the throwing motion is controlled by a spiking neural network. The networks topology is illustrated in Figure 6. It has three input and seven output neurons which are fully connected. The three dimensional position of the cylinder is used for the input neurons. Six of the output neurons control the six joints of the robot arm and the last neuron tells the hand when it should release the cylinder.

The absolute value of the cylinders position is directly set as the amplitude of the input neurons. This results in the spike rate of the neurons increasing as the absolute position values of the cylinder increase. The spike rate of the first six output neurons is passed to the arm joints as a position which produces a movement of the arm. The spikes of the seventh output neuron control the release of the cylinder, optimally resulting in the cylinder flying away.

A typical spike train of the input neurons is depicted in Figure 7. There aren't many spikes in the *Approach* and *Grasp* states as the cylinder is on the table and close to the origin. As

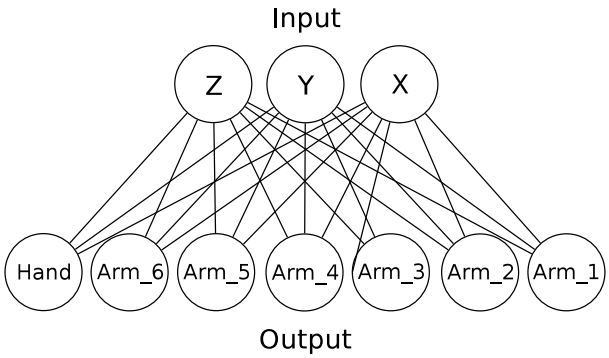


Fig. 6. The spiking neural network takes the cylinder coordinates $X, Y, Z$ as inputs and has seven outputs controlling the robot arm configuration and the release point.

the robot reaches back, first the $Z$ and then the $Y$ coordinate gets bigger and the corresponding neurons starts firing. At the end, the neural network takes control over the arm to execute the throwing motion. The spikes now depend on the motion the net dictates the robot as the cylinder is still in the hand. Afterwards, as soon as the cylinder leaves the hand, the spike rates of all input neurons would increase as the cylinder flies away.

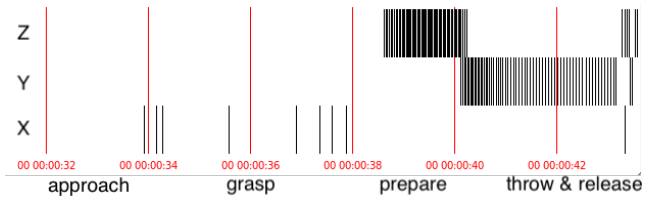


Fig. 7. The network spikes depending on the cylinder position input values. The higher the value, the more the spike rate increases. For instance, the neuron spikes more during the preparation state, because the arm is moving the cylinder up.

For the evolutionary algorithm, each individual is a list of 21 values, each representing one weight of the network. To evaluate the individuals, a throw is simulated and the distance of the cylinder from the table is measured, which represents the fitness of the individual. After each individual of a generation is evaluated, the elite, which consists of the best 50% of the individuals, is selected. The elite gets copied into the next generation and the remaining individuals are generated by mutating and recombining the individuals of the elite.

*1) Mutation Strategies:* Three different mutation strategies are implemented to generate new descendants. The first method uses only one parent and adds or subtracts small random values from its weights to produce an offspring. The other two methods are a single-point crossover and a k-point crossover as described in section II.

### B. Simplified Problem

As the first learning approach isn't very successful and it is learning very slowly, a reduction of the number of parameters and thus the search space for the evolutionary algorithm may

be beneficial. To achieve this, some joints are excluded from the throwing motion. All rotational joints are fixed by setting the respective weights in the neural network to zero, because these probably won't impact the robots ability to make a good throw very much. This reduces the number of weights to be learned from 21 to 12.

## VI. Problems

While working on the challenge, some problems arose with the neurorobotics platform itself as well as the server infrastructure around it. The simulations in the neurorobotics platform are non-deterministic which leads to non-reproducible results. This non-determinism probably results from floating point inaccuracy. In addition to that, the grasping of the cylinder is very unreliable. If the cylinder is grasped too tightly the robot starts shaking, sometimes resulting in the hand exploding and the cylinder flying huge distances (see Figure 8). In some extreme cases, the whole table the robot is mounted to moves, which results in the grasping not working afterwards due to a change in the relation of the robot and world coordinate systems. There were also problems with the platform crashing regularly because of different reasons. All these problems occurred on the server install of the platform. In absence of a working local install, it wasn't possible to check if they were specific to the server install or the platform itself.

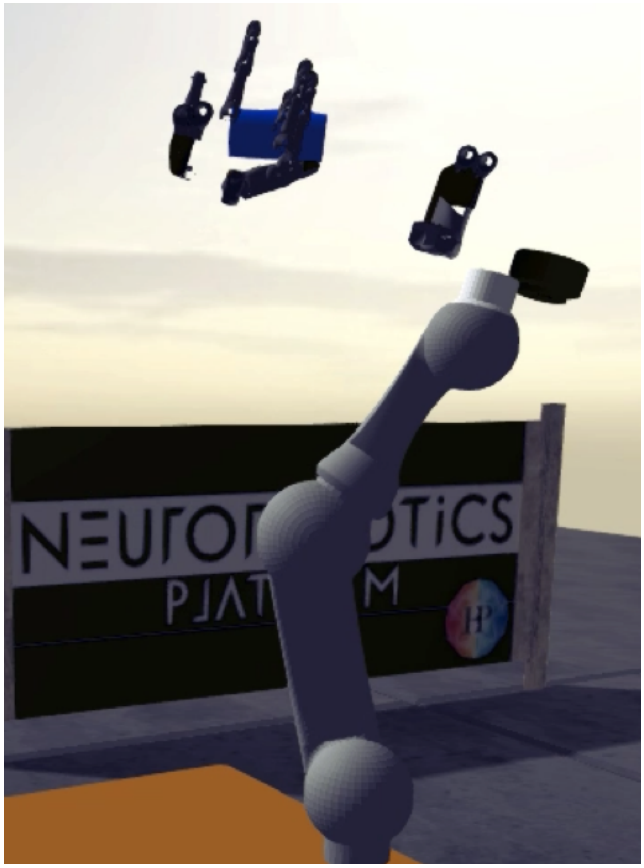

Fig. 8. Occasionally, different parts of the robot scatter.

## VII. Results

In the first approach, the robot is enabled to grasp the cylinder at random positions on the table and to throw it using a set of empirically determined parameters.

The second approach tries to learn a set of synaptic weights using an evolutionary algorithm. Figure 9 shows a plot of 10 generations with 6 individuals of the evolutionary algorithm. Depicted are the maximum, minimum and mean distances achieved in each generation. There is a slight upward trend noticable. However, the problems described in section VI make it very hard to learn anything. Therefore no propper learning can be observed.

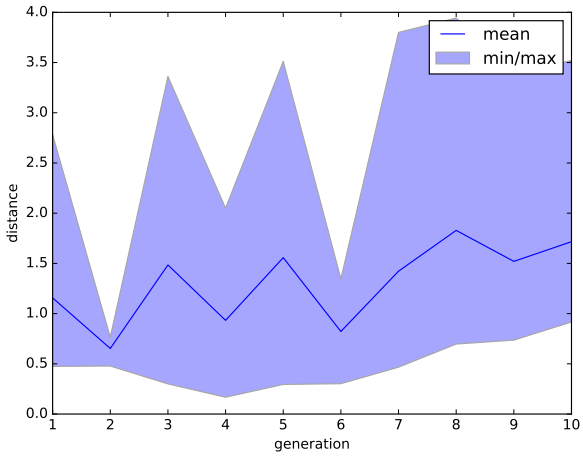


Fig. 9. 10 generations with 6 individuals of the evolutionary algorithm.

The individuals of the evolutionary algorithm can execute throws, but due to glitches in the simulation, like the shaking and exploding hand (Figure 8), the measured distances are not representative for the quality of the movement. In addition, the non-determinism in the simulation makes it even harder to compare the results as the same individual gets different distances when executed multiple times. This can be observed by the dips in the maximum curve. These issues result in the algorithm not really learning anything and evolving the weights rather randomly. Finally, the crashing platform doesn't allow to run the algorithm for sufficient generations.

## VIII. Conclusion

Two different approaches are presented to make the robot arm throw the cylinder. The first approach controls the robot arm with hard-coded configurations which have been determined empirically. This not only involves the static movements of the robot, but also grasping the cylinder at random positions reliably, which is computed with inverse kinematics. The second approach aims for a learned solution based on spiking neural networks. For this purpose, an evolutionary algorithm has been chosen in order to optimize the synaptic weights. Given the circumstances, various approaches have been implemented that lead to reasonable results. However,

the results are not comparable due to the non-deterministic behavior of the simulation platform.

### A. Future Work

Having set a basis for learning the networks parameters, different population sizes and numbers of generations with different mutation strategies could be compared in the evolutionary algorithm. Moreover, finding a good set of initial synaptic weights could help to converge faster towards a desired solution. However, unconventional solutions, such as fast and wild spinnings of the robot, that don't look like natural and realizable throwing movements, may, in that case, not be among the solutions. To further reduce the number of parameters, it could be considerable to use only the $y$-coordinate of the cylinder as an input, assuming that the cylinder is thrown in $y$-direction. In order to not only learn the throwing sequence, the grasping part could also be learned. The learning process in general could be supported by other inputs such as cameras or haptic sensors, but these would again increase the number of parameters to be learned. Finally, artificial neural networks could be used for learning in comparison to evolutionary algorithms.

## REFERENCES


[1] R. Dillmann, J. M. Zöllner, "Machine Learning 1 - Basic Methods", lecture 14, 2017

[2] A. Tavanaei, M. Ghodrati, S. Kheradpisheh, "Deep Learning in Spiking Neural Networks", 2018

[3] F. Ponulak, A. Kasinski, "Introduction to spiking neural networks: Information processing, learning and applications.", 2011

[4] Z. Bing, C.Meschede, F. Röhrbein, "A Survey of Robotics Control Based on Learning-Inspired Spiking Neural Networks", 2018