HyperNEAT Controlled Robots Learn How to Drive on Roads in Simulated Environment

Jan Drchal, Jan Koutník, Miroslav Šnorek

Abstract— **In this paper we describe simulation of autonomous robots controlled by recurrent neural networks, which are evolved through indirect encoding using HyperNEAT algorithm. The robots utilize 180 degree wide sensor array. Thanks to the scalability of the neural network generated by HyperNEAT, the sensor array can have various resolution. This would allow to use camera as an input for neural network controller used in real robot. The robots were simulated using software simulation environment. In the experiments the robots were trained to drive with imaximum average speed. Such fitness forces them to learn how to drive on roads and avoid collisions. Evolved neural networks show excellent scalability. Scaling of the sensory input breaks performance of the robots, which should be gained back with re-training of the robot with a different sensory input resolution.**

I. Introduction

In autonomous agents sensory input processing plays a crucial role when designing and implementing their controllers. Artificial agents – robots are faced to continuous real world environment through sensors like cameras, radars etc. with possibly high resolution in space and time domain. Typical processing of sensory input involves discretization and preprocessing for the robot control system.

Our goal is to generate robotic controllers based on recurrent artificial neural networks trained with evolutionary algorithm. Recurrent neural networks [1] are capable of effective temporal information processing because feedback connections form a short term memory within the networks. Such controllers can express more complex behavior. Second goal is to develop large-scale neural networks that can be directly used to process high dimensional sensory inputs. Afterwards, it will be possible to use camera images as neural network inputs, without massive preprocessing using computer vision algorithms.

There are many options how to transform preprocessed sensory input to actions that the robot performs in order to fulfill goals. Artificial neural networks can play the role of such a controlling system. In artificial neural networks the dimensionality of the sensory input was the obstacle that blocked direct processing of e.g. camera images. To overcome this limitation, we use hypercube encoding of neural network weights [2], which allows to increase input vector as well as amount of artificial neurons in the networks.

Hypercube encoding allows the large-scale neural networks to be effectively encoded into population of individuals. A single genome size does not grow with the number of neurons in the network. Similarly, resolution of the network inputs can be extended without growth of the network genome. This is the property of HyperNEAT algorithm used.

HyperNEAT algorithm was introduced in [2] and [3]. It is an evolutionary algorithm able to evolve large-scale networks utilizing so called generative encoding. HyperNEAT evolves neural networks in a two step process: the NEAT (see below) is used to create networks combining a set of transfer functions into a special function. The transfer functions allow to encode symmetry, imperfect symmetry and repetition with variation. The composed functions are called the Compositional Pattern Producing Networks (CPPNs). In the second step, planned neurons are given spatial coordinates. The previously evolved CPPN is then used to determine synaptic weights between all pairs (or subset of pairs) of neurons. The coordinates of both neurons are fed into the CPPNs inputs, the CPPN then outputs their connection weight. The weight is not expressed if its absolute value is below a given threshold. Such connectivity pattern created by CPPN is called the substrate. The important feature of HyperNEAT substrate is that it can be scaled to higher resolutions approximately preserving its inner structure and function.

NEAT (NeuroEvolution of Augmenting Topologies) [4] is an algorithm originally developed for evolution of both parameters (weights) and topology of artificial neural networks. It was extended to produce the CPPNs in the HyperNEAT algorithm instead of producing the neural networks directly. It works with genomes of variable size. NEAT introduced a concept of historical markings, which are gene labels allowing effective genome alignment in order to facilitate crossover-like operations. Moreover, historical markings are used for computation of a genotypical distance of two individuals. The distance measure is needed by niching evolutionary algorithm, which is a core of the NEAT. Because NEAT evolves networks of different complexity (sizes) niching was found to be necessary for protection of new topology innovations. The important NEAT property is the complexification – it starts with simple networks and gradually adds new neurons and connections. For evolving CPPNs, NEAT was extended to evolve heterogeneous computational units (nodes).

A. Related Work

Evolution of artificial neural networks is a robust technique for development of neural systems. Many techniques were developed for evolution of either weights or even a structure of neural networks like e.g. Analog Genetic Encoding [5],

Jan Drchal, Jan Koutník and Miroslav Šnorek are with Computational Intelligence Group at the Department of Computer Science and Engineering at the Faculty of Electrical Engineering at Czech Technical University in Prague (phone +420-22435-7470; fax: +420-22492-3325 e-mail: {koutnij|drchal|snorek}@fel.cvut.cz, http://cig.felk.cvut.cz)

[6], [7], Continual Evolution Algorithm [8], GNARL for recurrent neural networks [9], Evolino [10] and NeuroEvolution of Augmenting Topologies (NEAT) [4]. The NEAT algorithm became a part of HyperNEAT algorithm as a tool for evolution of CPPNs.

HyperNEAT algorithm was already applied to control artificial robots in food gathering problem [2] A robot with a set of range-finder sensors is controlled to approach the food. It was shown that HyperNEAT is able to evolve very large neural networks with more than eight million connections. A very interesting property of the HyperNEAT is the ability to change the resolution of the substrate. For example 11×11 grid was resized to 55×55 while preserving the underlying neural network function. In the food gathering experiment the inputs indicating whether the food is in the particular direction were arranged parallel or concentric with the robot body. Each sensor was geometrically linked with an effector, which drives the robot in the particular direction.

Our approach differs in the organization of the input sensors, which are arranged in polar rays having particular angular and distance resolution. The sensors are sensitive to color of the surface and in fact represent a camera with arbitrary pixel resolution.

In [11] HyperNEAT algorithm was applied in a very efficient way so that each agent shares a portion of the substrate and neural network. The neural network splits to local areas in substrate geometrically but all agents share one substrate. This can be exploited in agents cooperative behavior.

In [12] it is shown that robots can complete common goals with a minimum information coming from sensors. The robots are controlled with evolved feed-forward neural networks.

This paper is organized as follows. In the next section our approach to evolution of robotic controllers and their testing is described. Section 3 describes the simulation environment and the robot setup. Section 4 describes the experimental results. Final section concludes the paper.

II. Our Approach

In our approach, we reduced effort typically required to build hardware robotic platforms such as described in [12]. We moved directly to the simulation to concentrate on development of the robots control algorithms. First, we created a simulation environment described in Section III-A. The environment allows rapid development and experimentation with simulated robots.

The experiment in this paper reflects widely solved task to create a robotic controller, which controls mobile robot to drive and stay on road. In the first experiment, the robots do not fulfill any other goal such as food gathering, maze exploration etc. but such experiments can be easily performed in the simulation environment. We do not pay attention where the robots drive to. This is not a typical task but we can study the emerged robots behavior. For example, we expect the robots to learn how to drive on one side of the road to avoid collisions.

Fitness function in experiments is an average speed of the simulated robots. The speed is meaningless variable (number of pixels per a simulation step) but can be computed in a straightforward way and is suitably proportional.

In the simple experiment we also did not pass information about other robots through the sensors to the neural network controllers. The robots can hit each other, which slows them down and decreases their fitness value.

All robots have different instances of the same robotic controller. This means that state stored in the recurrent neural network is a property of the simulated robot instance. The co-evolution experiment can be realized as well.

The input substrate described in Section III-B and Figure 2 is an array of real values in interval between 0 and 1. The road surface (black) is encoded by 1 and the grass (green) is encoded by 0.

III. Experimental Setup

A. Simulation Environment

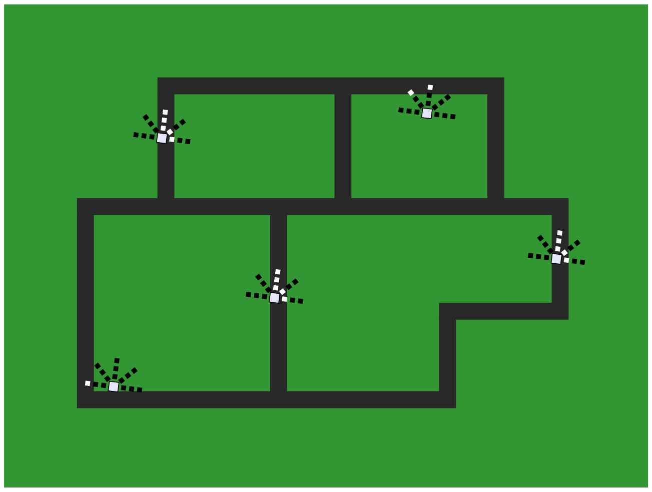


Fig. 1. Experimental scenario with robots placed in starting positions. Each robot in the example has 3x5 sensors. The color of the sensor represents a surface friction mapped to 0 and 1 for the neural network input.

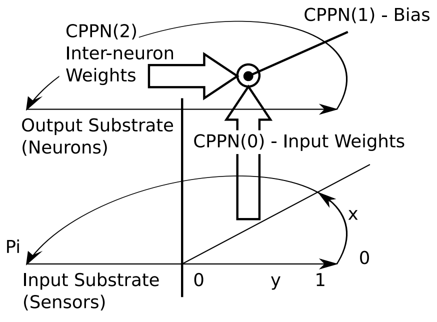
For simulation of the robots a new simulation environment (named ViVAE - Visual Vector Agent Environment) was built. The environment is written in Java. Configuration of the input scenario is done through SVG standard. The simulation scenario is simply drawn in an SVG vector editor or the SVG file can be (e.g. randomly) generated. There is a set of objects that can be used as surfaces with a different friction coefficients, movable and fixed obstacles as well as robots. The robots can be equipped with sensors that recognize objects directly. This approach can reduce the problem with recognition of objects in sight. The sensors receive objects in queues. The sensors for surface friction receive an object under the sensor from which its friction is read.

The environment supports basic physics and resolves conflicts between objects. It does not respect elasticity of the objects and objects are not deformable.

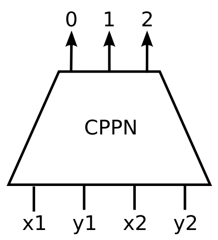
The simulation environment allows easy design of a new experiment and easy snapshoting of the current simulation state including its visualization. Simulator loads the SVG with all objects in the simulation, loads the classes that represent the objects and starts the simulation. In every step, the simulation state can be saved back to the SVG format and used for further visualization or experiments.

B. Robot Setup

All robots performing in the scenario are equipped with the same neural network controller generated from a single individual in the evolutionary algorithm. Opposite approach can utilize co-evolution of the robots or can exploit some relationship among the robots in the population and evolve robots that exhibit e.g. altruistic behavior [13].



(a)



(b)

Fig. 2. Organization of the HyperNEAT substrate. There were two distinct substrates used (a) and the CPPN has three outputs. CPPN(0) output is a weight between input substrate (sensor) and a neuron in the upper substrate. Second, CPPN(1) output is used as bias for neurons in the upper substrate. For bias calculation third and fourth CPPN inputs are set to 0. Last CPPN(2) output represents connection weights among neurons in the upper substrate.

Simulated robot is driven by two simulated wheels and is equipped with a number of sensors. The robot is controlled by neural network. The neural network is organized in a single layer of possibly fully interconnected perceptron (global) type neurons (neurons compute biased scalar product, which is transformed by bipolar logistic sigmoidal function). Steering angle is proportional to inverse actual speed of the robot.

The sensors as well as the neural network are spread in a HyperNEAT substrate. Neurons and sensors are addressed with polar coordinates, see Figure 2. Two of the neurons in the output substrate are dedicated to control acceleration of the wheels. The neuron activation is the corresponding wheel acceleration. The neurons are marked with the red color in Figure 3.

The CPPN function has three outputs that express weights between input substrate (sensors) and the output neurons (output 0), bias for the neurons (output 1) and weights among the output neurons (output 2). Since the bias is a property of a neuron in the output substrate, such CPPN output is a function of two variables (x_1 and y_1). The other two variables (x_2, y_2) were set to zero when computing the biases.

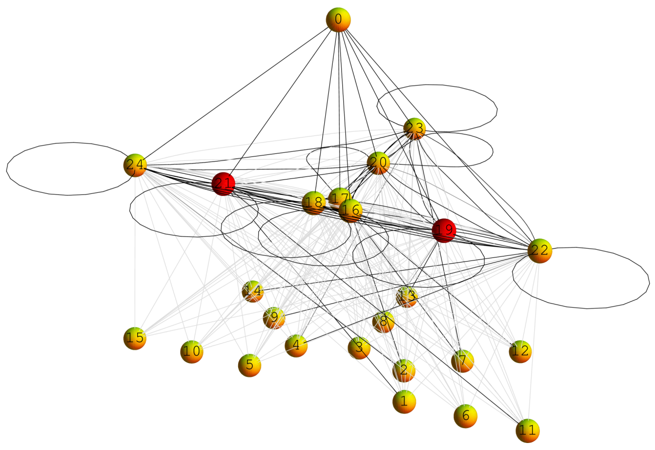


Fig. 3. Visualization of the evolved neural network controlling the robots. The figure contains two layers. The bottom layer represents input sensors in a grid of 3x5 inputs. The upper layer contains 9 (3 × 3) neurons. The neurons are mapped into a substrate in polar coordinates to match shape of the input substrate. Red spheres represent neurons that steer the robot wheels. The most upper sphere represents bias for the neurons. Connections are displayed with lines. The most visible lines represent connections with stronger synaptic values.

C. HyperNEAT setup

We have used our own implementation of the HyperNEAT algorithm. The NEAT part resembles Stanley's original implementation. The HyperNEAT extension is inspired mainly by the David D'Ambrosio's HyperSharpNEAT[1]. Table I shows CPPN node functions.

TABLE I
CPPN NODE FUNCTIONS

Name	**Equation**
Bipolar Sigmoid	$\frac{2}{1+e^{-4.9\,x}} - 1$
Linear	x
Gaussian	$e^{-2.5\,x^2}$
Absolute value	$\lvert x \rvert$
Sine	$sin(x)$
Cosine	$cos(x)$

The parameter settings are summarized in Table II. Note, that we have extended the original set of constants which determine the genotype distance between two individuals (C_1, C_2 and C_3) by the new constant C_{ACT}. The constant C_{ACT} was added due to the fact that, unlike in classic NEAT, we evolve networks (CPPNs) with heterogeneous nodes. C_{ACT} multiplies the number of not matching output nodes of aligned link genes. The CPPN output nodes were limited to bipolar sigmoidal functions in order to constrain the output.

IV. Experimental Results

In Figure 1, we can see an experimental scenario with five robots placed at their starting positions. All robots are

[1]Both Stanley's original NEAT implementation and D'Ambrosio's HyperSharpNEAT can be found on `http://www.cs.ucf.edu/~kstanley`.

TABLE II

HYPERNEAT PARAMETERS

Parameter	Value
population size	100
CPPN weights amplitude	3.0
CPPN output amplitude	1.0
controller network weights amplitude	3.0
distance threshold	15.0
distance C_1	2.0
distance C_2	2.0
distance C_3	0.5
distance C_{ACT}	1.0
mating probability	0.75
add link mutation probability	0.3
add node mutation probability	0.1
elitism per species	5%

equipped with the same neural controller.

The following text describes a typical evolutionary algorithm run. Figure 4 shows trajectories of robots driven by the best controller of the first generation, where the population is completely randomized. It can be seen despite the fact it was the first generation, the best controller was already able to keep robots close to the road where the movement is much faster.

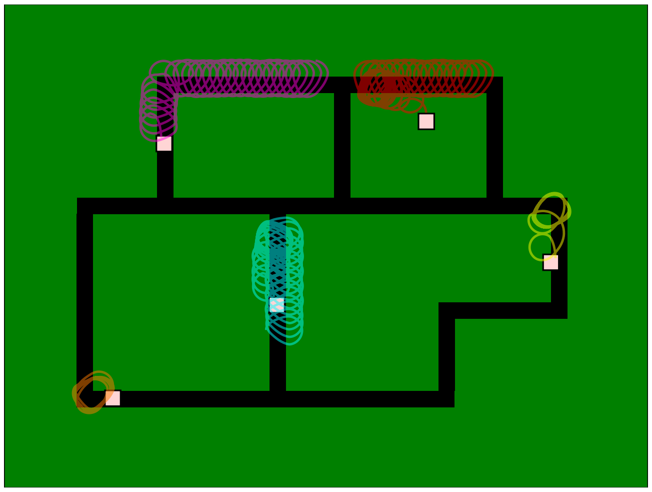


Fig. 4. Trajectories of the robots controlled by the best of neural networks generated by a CPPN randomly generated in the first generation of the evolution. Fitness = 1.505.

Figure 5 shows the situation in the 10th generation. Robots now almost perfectly avoid the grass. They still loose speed by instant steering.

In the generation 252 a strategy to drive along roads was found. The trajectories are shown in Figure 6. Unfortunately, robots were able to turn to one side only. They realized the turn in the opposite direction by turning by 270 degrees. Such maneuver of course reduced their average speed.

Figure 7 shows the final solution which was found in the

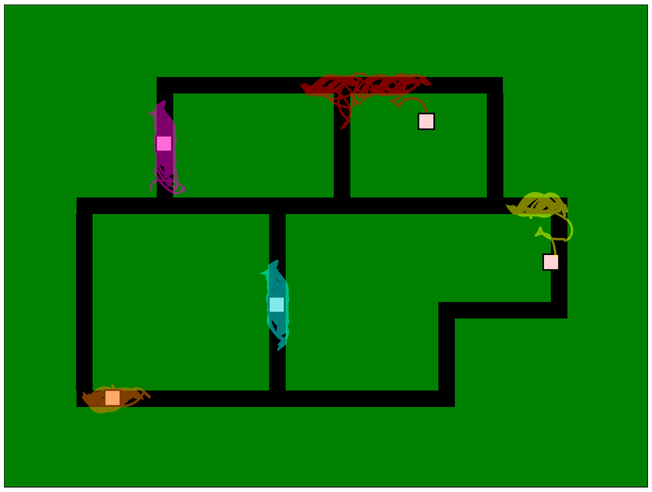


Fig. 5. Trajectories of the robots produced in the 10th generation. Fitness = 3.558.

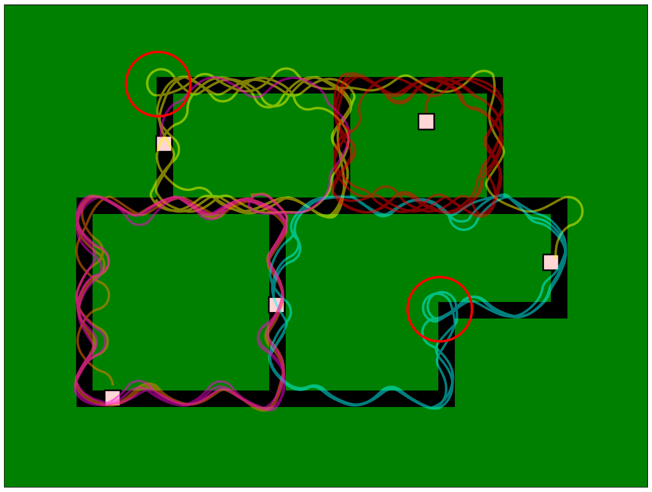


Fig. 6. Trajectories of the robots controlled by a neural network found in generation 252. The robots are able to stay on the road and drive in one direction. The robots are able to turn to one side only, other turns are made by turning by 270 degrees to the other side as emphasized by red circles. Fitness = 3.865.

generation 324. The trajectories are fluid. Moreover, robots learned to drive on one side of the road to avoid mutual collisions.

Figure 3 depicts the neural controller of the final solution. Here, 223 out of possible 225 connections were expressed.

Finally, Figure 8 displays typical convergence of the HyperNEAT evolution in 400 iterations. Each generation of individuals in the is sorted. We can see that diversity among the population is maintained.

A. Substrate Resolution

As we have already mentioned, HyperNEAT method is known to be able to preserve the functionality of the evolved networks even after the substrate density is increased. Figure 9 shows trajectories of robots driven by champion controller from the previous experiment with doubled density of the output substrate. The network was not retrained with

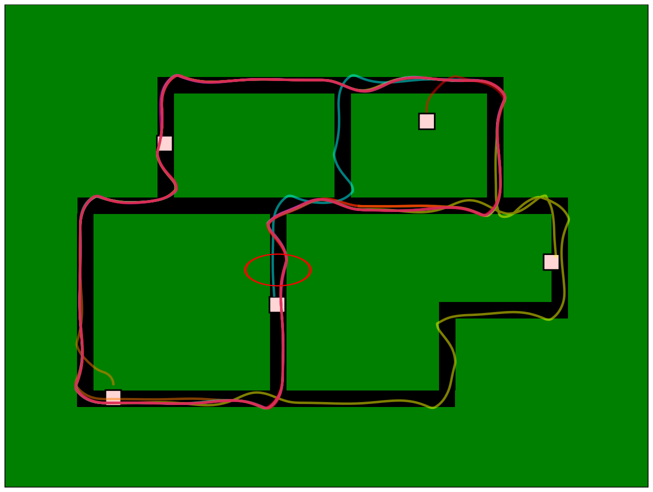


Fig. 7. Trajectories of the robots controlled by the neural network found in generation 324. The trajectories are fluid. The robots learned how to drive on one side of the road (as emphasized by the red ellipse). Fitness = 4.256.

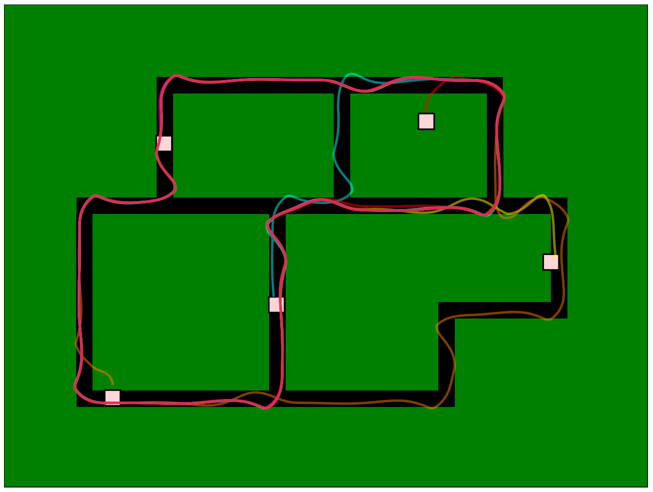


Fig. 9. Trajectories of the robots controlled by the neural network champion of generation 324. The density of the output substrate was doubled to 9×9 (81 neurons). Fitness = 4.243. No major degradation was encountered, compare with Figure 7.

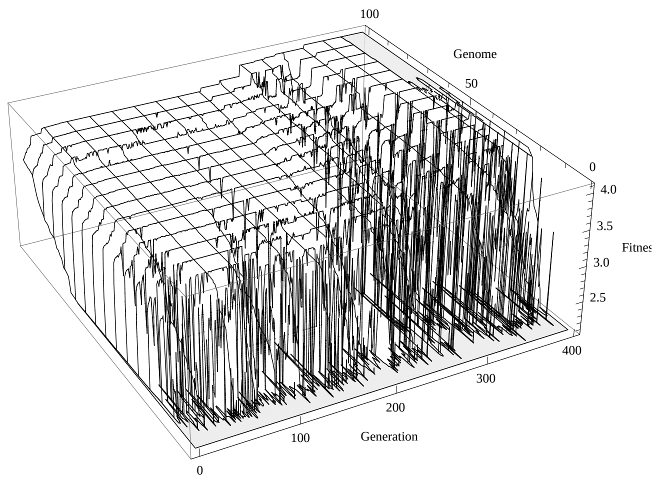


Fig. 8. Typical convergence of the HyperNEAT algorithm in 400 generations. Each generation of 100 individuals is sorted and displayed. We can see, how the diversity in populations is maintained.

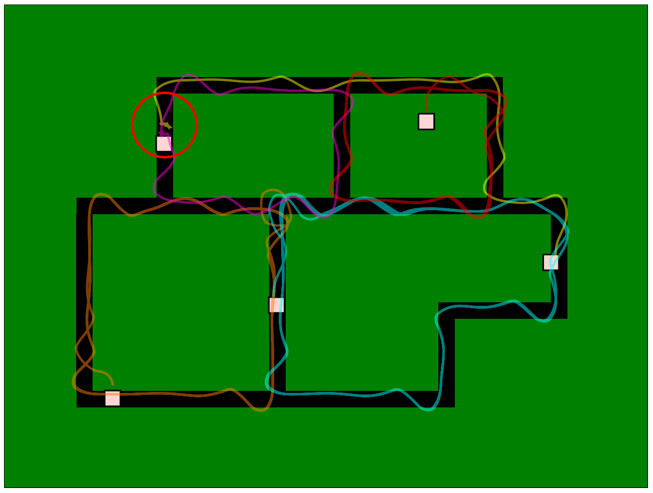


Fig. 10. Trajectories when denser input substrate 3×6 was used. Fitness = 2.792. Note, the red circle which marks the spot where two robots collided and became stuck. The method is more sensitive to changes of the density of the input substrate than the output substrate.

the new resolution in the substrate. The new controller has $9 \times 9 = 81$ neurons in the output substrate which is nine times more. Still, no visible changes were encountered, compare with Figure 7.

In the next experiment we have increased the density of the input substrate to $3 \times 6 = 18$ neurons. The result is shown in Figure 10. One can see, that the performance dropped, this was mainly caused by the collision of two robots. However, preliminary experiments show that only few evolutionary steps are needed to correct these errors, which confirms the Stanley's results [3].

V. Conclusion and Future Work

In this paper, we present experiments with neural network controlled robots. We used HyperNEAT method which is able to evolve large-scale neural networks. This is possible due to its usage of generative encoding. We employed simulated 2D physical environment ViVAE, which facilitated fast design and simple modifications to robot evaluation scenarios. Robots process surface color information (which represents the surface friction) with sensors organized in polar coordinates in two quadrants in front of the robot. Extension to omni-directional camera should be straightforward.

The robot artificial neural network controllers were trained to maximize robot's average speed in the arena. The correct strategy was always found: robots discovered a way how to drive on roads avoiding grass terrain, which slowed them down. Moreover, they learned how to drive on one side of the road effectively avoiding mutual collisions.

We have found that HyperNEAT method is more resistant to the scaling of the output substrate than to the scaling of the inputs. Our present preliminary experiments show that only a limited number of evolutionary steps is needed to correct controllers with up-scaled substrates. Also, we expect that change of the substrate density during a single evolutionary run can generalize the underlying CPPN in a way that scaling to higher densities will be more feasible.

In the future, we plan experiments using real, not simulated, robots using omni-directional cameras. Therefore thorough experimentation with increasing of input substrate has to be done.

We also plan to use another input substrate for encoding of fixed obstacles and other moving robots. This will allow to create and run more complex experiments.

Acknowledgment

The authors would like to thank Zdeněk Buk who created the neural network visualization in Wolfram Mathematica. As well as CTU student Petr Smejkal who created the first implementation of the robots simulation environment. This work has been supported by the research program "Transdisciplinary Research in the Area of Biomedical Engineering II" (MSM6840770012) sponsored by the Ministry of Education, Youth and Sports of the Czech Republic.

References

[1] J. L. Elman, "Finding structure in time." *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[2] D. B. D'Ambrosio and K. O. Stanley, "A novel generative encoding for exploiting neural network sensor and output geometry," in *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2007, pp. 974–981.

[3] J. Gauci and K. Stanley, "Generating large-scale neural networks through discovering geometric regularities," in *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2007, pp. 997–1004.

[4] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, pp. 99–127, 2002.

[5] C. Mattiussi, "Evolutionary synthesis of analog networks," Ph.D. dissertation, EPFL, Lausanne, 2005. [Online]. Available: http://library.epfl.ch/theses/?nr=3199

[6] P. Dürr, C. Mattiussi, and D. Floreano, "Neuroevolution with Analog Genetic Encoding," in *Parallel Problem Solving from Nature - PPSN iX*, ser. Lecture Notes in Computer Science, vol. 9, 2006, pp. 671–680. [Online]. Available: http://ppsn2006.raunvis.hi.is/

[7] P. Dürr, C. Mattiussi, A. Soltoggio, and D. Floreano, "Evolvability of Neuromodulated Learning for Robots," in *The 2008 ECSIS Symposium on Learning and Adaptive Behavior in Robotic Systems*. Los Alamitos, CA: IEEE Computer Society, 2008, pp. 41–46.

[8] Z. Buk and M. Šnorek, "Hybrid evolution of heterogeneous neural networks," in *Artificial Neural Networks - ICANN 2008*, vol. 5163. Springer Berlin / Heidelberg, 2008, pp. 426–434.

[9] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, pp. 54–65, 1993.

[10] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez, "Training recurrent networks by evolino," *Neural computation*, vol. 19, no. 3, pp. 757–779, March 2007.

[11] D. B. D'Ambrosio and K. O. Stanley, "Generative encoding for multiagent learning," in *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2008, pp. 819–826.

[12] M. Waibel, "Evolution of cooperation in artificial ants," Ph.D. dissertation, EPFL, 2007.

[13] D. Floreano, S. Mitri, A. Perez-Uribe, and L. Keller, "Evolution of altruistic robots," in *Proceedings of the WCCI 2008*, vol. 5050. Springer Berlin / Heidelberg, 2008, pp. 232–248.