C. Aykın, M. Knopp, and K. Diepold, "Deep Reinforcement Learning for Formation Control", 2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), Nanjing, China, 2018, pp. 1124-1128. DOI: 10.1109/ROMAN.2018.8525765

Deep Reinforcement Learning for Formation Control

Can Aykın, Martin Knopp[1], and Klaus Diepold

***Abstract*— Continuing our work on using reinforcement learning for formation control, we present an end-to-end deep learning system which uses only camera images to learn to control the individual system's correct position within the formation.**

Mnih et al. created AIs playing video games utilizing the same visual input as a human player by employing convolutional neural networks for automatic feature extraction on images. This published work inspired us to employ a similar approach for processing the camera images and controlling the robot.

We repeat the same experiment with two completely different camera positions. The results for both positions are very similar and such demonstrate the flexibility of the presented approach.

I. Introduction

We are interested in solving the problem of formation control using reinforcement learning algorithms. With this approach, we want to create a flexible framework that can autonomously adapt to varying environment conditions and to different agents. We hope to solve problems like formation control of robotic wheelchairs, which are accompanied by human caregivers [1], or multi-robot formations in shared rescue missions [2] using semi-automatic training instead of an individually tailored solution.

In our previous work [3], we investigated whether reinforcement learning can be used to let a follower learn to stay in line formation behind its leader. We solved this by employing the GQ(λ) reinforcement learning algorithm. We also introduced an update buffering technique, which allowed a certain degree of asynchronicity between the learning algorithm and the execution of the robots' commands. This technique yields significant performance improvements in simulations and is important for real applications as we have to carefully control timing between different actions and changes of the environment and the learning algorithm itself otherwise.

Our approach also showed some issues, the most important one being that our solution had to carefully observe the specific properties of the considered robots. This becomes most obvious in the state-space we used. Directly using the values of the proximity sensors creates a state-space that cannot be processed anymore (4096^2). Therefore, we had to map the raw values from the proximity sensors onto a lower dimensional space. The projected values are than used as an input to our algorithm. The problem is that this mapping already includes knowledge about the meaning of values (e.g. linearity of the sensors, correspondence of values to concrete areas of observation, all possible observation areas have the same size) and it heavily depends on the used distance sensors and the concrete formation we investigated.

Keeping this in mind, we require two characteristics for a new solution: first, it has to either process a high dimensional state-space or infer a suitable lower-dimensional representation on its own. And second, it has to adapt to changes in the environment in a flexible way without requiring manual adjustment of parameters.

Inspired by the work of Mnih et al. ([4], [5]), we present an end-to-end deep reinforcement learning approach for formation control that does not depend on specific distance sensors, but uses conventional camera images to learn how to control its movements in order to stay in the desired formation. In contrast to our previous work, the state-space is not hand-crafted for the problem at hand, but approximated internally by a *Deep Q-learning Network (DQN)*.

In the next section, we introduce the specific properties of DQNs that make them suitable for creating end-to-end control systems and how they solve specific problems, which are usually problematic when trying to employ reinforcement learning on a task like ours.

As an explanation of reinforcement learning itself is far beyond the scope of this paper, we stick to explaining the ideas on a conceptual level, which we hope is comprehensible for readers both familiar and unfamiliar with reinforcement learning. For a thorough explanation of the ideas, concepts, and algorithms of reinforcement learning, we refer to Richard Sutton's book [6] on this topic.

After these considerations, we look at the specific formation control experiments we did and how they are performed by such an end-to-end deep learning approach.

II. Deep Q-Learning

In recent years, big improvements in computer vision, speech and text processing (e.g. [7], [8], [9], [10], [11]) have been made possible by the usage of deep neural networks and similar techniques which can utilize a lot of parallel computations. They are extremely powerful in regard to feature extraction and classification. Mnih et al. [4] used such a classification network, more precisely a *Convolutional Neural Network (CNN)*, in the core of a framework playing five different Atari 2600 video games in a way very similar to a human player. That is, by reacting to images on the screen instead of depending on some knowledge about the internal state of a game which is what standard artificial opponents in video games are doing.

[1]Corresponding author: Martin.Knopp@tum.de

All authors are with the Department of Electrical and Computer Engineering, Technical University of Munich, 80333 Munich, Germany

In our previous work, we had to quantise the values of two distance sensors into a 9×9 grid, a common technique that is often referred to as *tile-coding*, to create a state-space which we then used to train a follower to stay within formation. The problem with this approach is that it is highly inflexible and has to be repeated for different robots, different sensors, or changing computational capabilities.

Mnih et al. also had to cope with an extremely large state-space (the dimension of the images) and handled this high dimensional state-space by training a CNN to directly approximate the *Q-function* (simplified: a mapping between states, actions, and rewards) out of images from the games. In contrast to common CNNs, the networks used for Q-learning do not contain max pooling layers, which are prevalent in classification tasks. In classification tasks, they provide a certain degree of translational invariance, which is very beneficial for classification, but unwanted for state representations. Applied to our case, we do not need information whether the player's avatar or the formation leader can be found in the image at all, but need to infer knowledge about their location in the image for the purpose of our algorithm.

Apart from efficiently coping with large state-spaces, the CNN approach also has another significant advantage: it is highly flexible regarding the composition of the image. Its single requirement is that information about the current state is somehow contained in the image. The algorithm decides by itself which parts of the image are important and which information can be gained out of them.

A straightforward approach on directly training the neural network to approximate the Q-function leads to many problems, the most important one being the huge correlation between consecutive states in a common reinforcement learning process. The huge correlations cause the learning algorithm to get stuck in local maxima and large variances in the updates occur in every time step. Often these problems become so severe that the learning algorithm diverges and does not learn anything useful at all.

To cope with these problems, Mnih et al. introduce *experience replay* in their algorithm, a technique, which stores previous tuples of state, action and reward into a replay buffer. Instead of using the current observed state, action, and reward to update the Q-function, a uniformly sampled episode from the replay buffer is used instead. This breaks the correlation between states and enables batch training of the neural network. It is also efficient regarding data usage as each episode gets used for multiple updates without requiring the agent to visit the same state again.

In our and Mnih's work, the replay buffer is implemented as a ring buffer, that is, when the replay buffer is full and new experience is added, the oldest one is removed from the buffer. This process does not distinguish between the importance of different samples to the learned knowledge. Assessing this importance would be very difficult, but fortunately, the simple ring buffer approach works well enough and is motivated directly from an implementation perspective.

A minor consideration of using random episodes instead of the current one is that it also limits the choice of the reinforcement learning algorithm to the class of *off-policy* algorithms as the learned policy is different to the one the agent currently executes. This is why Mnih et al. chose to employ *Q-learning*, a common off-policy algorithm.

Usually, the Q-function maps the current state-action pair to a certain reward, that is, it estimates the quality of a certain state-action pair. From a human perspective, an admissible state is the position of the player's avatar in a video game or the relative position of the leader to the follower in our formation control scenario. The idea behind Deep Q-learning is not to use these hand-crafted state representations but to take the raw sensor data (images) and let the neural network figure out on its own a feasible mapping from pixel values onto an internal state representation. The output of the network represents a quality assessment of every possible action the agent could execute. Due to the connection structures within the network, one training sample not only improves its own action assessment, but the complete prediction of all actions.

Using this Deep Q-learning network, we achieve an approximation to tell the agent which action is feasible to take depending on an image input. Similar to many other reinforcement learning processes, this information is utilized by an ϵ-greedy policy to balance between exploration and exploitation. Instead of the action with the highest expected reward, ϵ-greedy policies chose a random action with probability ϵ instead. This probability is reduced over the learning process, in our case linearly from an initial 0.4 to 0.01 over the first $100\,000$ steps. The reward estimates are initialized arbitrarily, so we want to ensure a good coverage of the entire state-space and we hence need a broader exploration in the beginning. After some time the estimates become more reliable and following the suggested action leads to a good reward.

Continuously updating the Q-network and using this information in the next time step to decide which action to take leads to an accumulation of errors and the divergence of the whole network [12]. To avoid this problem, the Deep Q-learning algorithm of Mnih et al. uses two networks: a slowly updated target network, which is only used in forward direction to calculate the current Q-values and the resulting loss of the network. And a primary Q-network, which is trained by this loss and whose parameters are copied onto the target network after a certain number of steps.

III. Scenario

This section describes the concrete scenario we investigated and what parameters we chose regarding the Deep Q-learning algorithm.

A. Environment, Robots, and Tasks

We consider *e-pucks* [13], small tabletop robots, which are simulated in V-REP [14], a physics simulator designed for robotics. Our simulation includes two robots: a leader following a predetermined path and a follower which is controlled by our learning algorithm. The first robot, the

leader, is statically configured to follow a line on the ground (Figure 1). Its speed is determined by multiplying a randomly

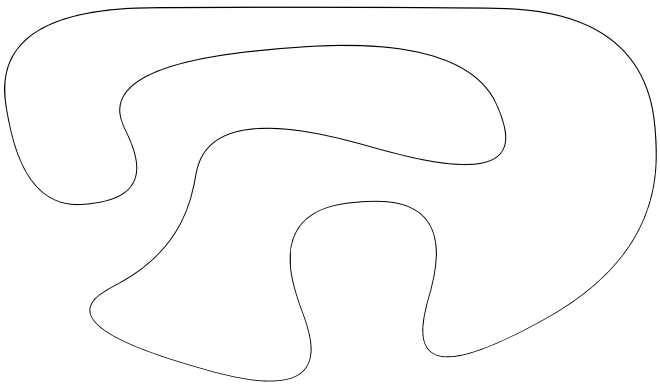

Fig. 1. The arbitrary path the leader uses in our experiments. It represents a non-trivial path which could be found in reality. Its closed-loop design allows the experiment to run continuously.

factor between -0.5 and 1 with the desired velocity. In this way the leader can also go backwards which makes the task harder to solve, but also more robust. This behaviour represents a non-trivial and realistic movement pattern which can also be found in reality like a human or a robot with SLAM based navigation planning.

The second robot is the designated follower we want to train. It has a forward-looking camera with an image angle of $45°$ and a resolution of 84×84 pixels. An example image from this camera can be seen in figure 2. The resolution of this camera seems very low at first glance, but larger resolutions increase the computational load dramatically and the results from using the same resolution as Mnih et al. did for Atari games are already very satisfying.

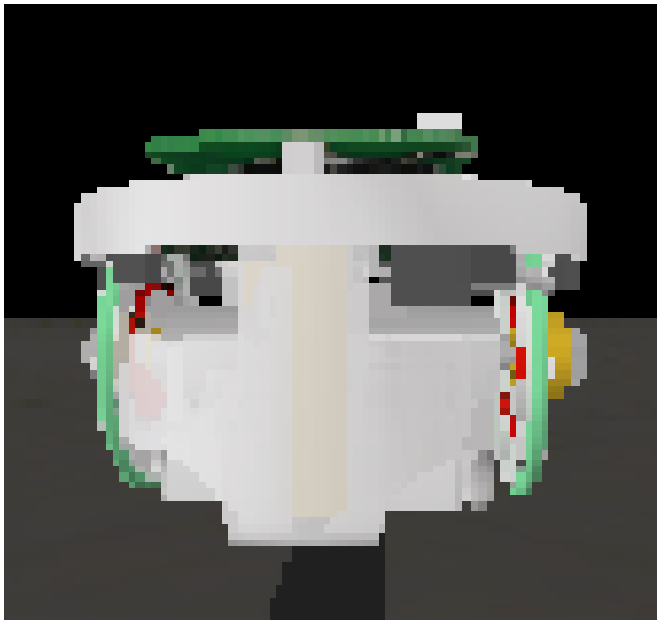

Fig. 2. One camera frame from our first trial: An 84×84 pixel view from the follower to its leader.

As our approach should allow end-to-end training without requiring changes to hyperparameters, we set up a second experiment to test this capability where we moved the camera to a bird's-eye position above the follower. This camera is placed about $55\,\mathrm{cm}$ above the robot (roughly 5 times the size of the robots themselves), looking downwards. An example of this view can be seen in figure 3.

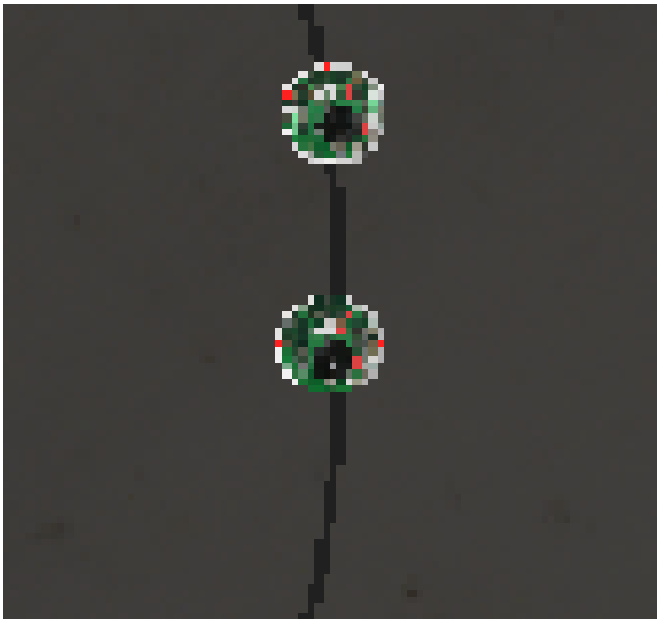

Fig. 3. One camera frame from our second trial: An 84×84 pixel view fixed above the follower.

The task of the second robot is to stay directly behind its leader at a distance of $15\,\mathrm{cm}$ (about three times its own radius). In each episode the robot achieves this goal, the learning algorithm receives a reward of $r = 1$. The following robot may deviate from this position by up to $6\,\mathrm{cm}$ and $20°$, resulting in a lower reward scaled proportionally to the error. If the second robot leaves this region, the current episode will end and a negative reward of -10 is given. It will then be moved back to the optimal position behind the leader ($15\,\mathrm{cm}$, $0°$) and a new episode starts. An illustration of the formation between both robots and the error measures can be seen in Figure 4.

B. Image Processing and Network Architecture

Before feeding the camera images into the network, we convert the RGB frames to greyscale to reduce the computational demands of the algorithm by a factor of three.

Changes of the leader's position are connected by a chain of movements of the leader. To allow the algorithm to learn these connections and base its decisions on velocity and trajectory while avoiding to feed inputs of arbitrary length into the network, we always take the last four consecutive frames to create one sample. The input to our neural network is therefore an array of four 84×84 greyscale images. Our neural network adheres to the original design in [5] and utilizes three hidden convolutional layers and two fully-connected layers. The first hidden layer convolves the input

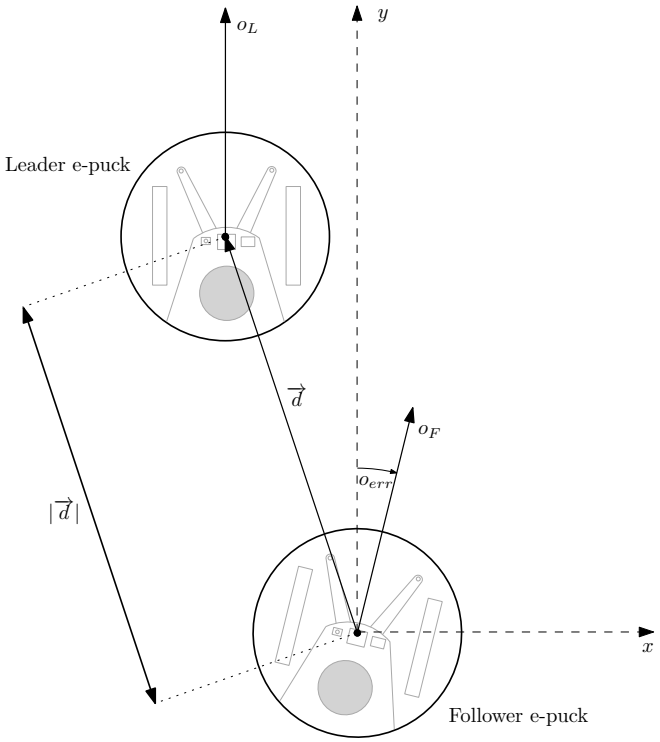

Fig. 4. Illustration of the error measures $|\vec{d}|$ and o_{err} for our leader-follower scenario (taken from our previous paper [3]).

images with 32 8×8 kernels with stride 4 and uses a rectifier activation function. The second one uses 64 4×4 kernels with stride 2 and also a rectifier function. The final convolutional layer uses 64 3×3 kernels with stride 1. The output is reshaped into an array and fed into a fully-connected layer consisting of 512 rectifier units. The mapping onto the 25 different actions is done by the second fully connected layer. Our 25 actions are all possible combinations of 5 different motor speeds for forward and backward movement as well as stopping and 5 different turn rates (chosen in a similar manner).

As described in the section about concepts of Deep Q-learning, we use two different networks to improve training stability. The parameter update from the primary Q-network onto the target network happens every 2000 steps.

Our network does not use batch normalization, which makes the batch size an important parameter. In our experiments, we use a batch size of 128 episodes for our initial tests in a synchronous environment and 256 episodes for the asynchronous environment presented here.

IV. Results and Discussion

The initial learning phase of the algorithm takes around 60 000 steps. We define the initial phase as the period where the algorithm regularly requires the start of a new episode, that is, it loses track of the leader and has to be put back into a starting position from where it can continue. After around 100 000 steps, the overall reduction of the error slows down significantly and converges asymptotically.

The errors of frontal and top-down view show a very similar development, which confirms the flexibility of the approach and which supports its applicability for end-to-end control directly from camera input to motor speeds.

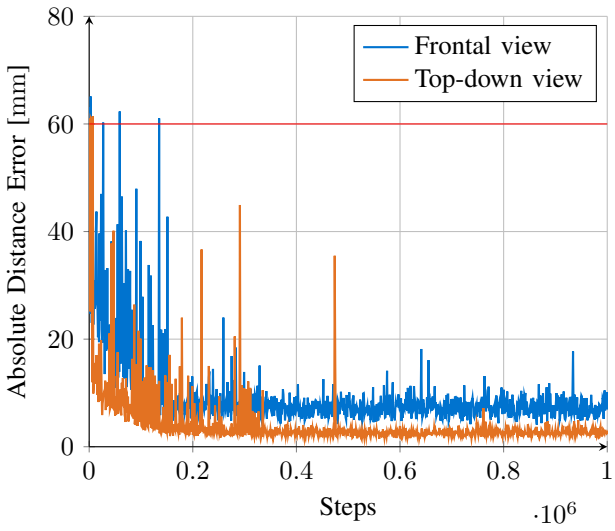

Fig. 5. Maximal distance error for frontal and top-down view over time. New episodes are started when the error surpasses 60 mm (red horizontal line). Convergence slows down significantly after 100 000 steps.

The remaining distance error averages around 1–2 mm, which is about 1 % of the required distance and 2 % of the size of the robots. The progression of the maximal encountered error during the simulation is shown in figure 5. The orientation error usually stays below 1° after the initial phase and the development of the maximal encountered orientation errors during the experiment can be seen in figure 6.

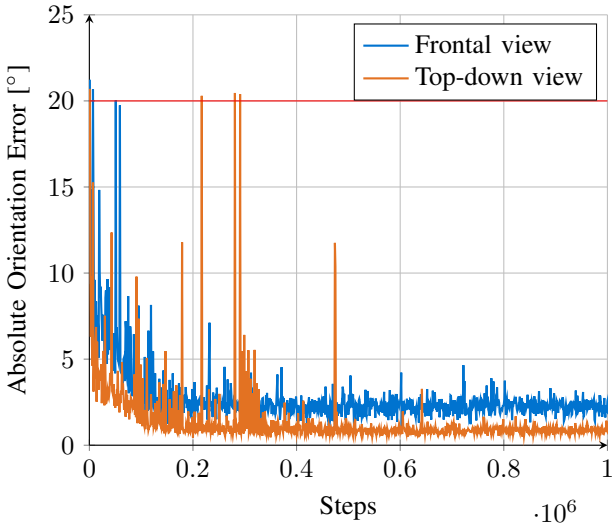

Fig. 6. Maximal orientation error for frontal and top-down view over time. Progression is very similar to the distance error. New episodes start when the error surpasses 20° (red horizontal line).

Although the top-down view allows for a lower overall error, it produces some spikes occurring up until 500 000

steps. The lower total error could be the result of the independence of position and orientation information in the top-down view. A left-right shift by a certain amount of pixels in the frontal facing image corresponds to different orientation errors depending on the size of the robot within the image. This relation does not exist in the top-down view, orientation and distance can be inferred independently of each other and the size of the robots in the image always stays the same.

On the other hand, this could be the reason for the observed error spikes, as the smaller size of both robots in the top-down image leads to a lower resolution of the inferred position and orientation information.

Another explanation for the errors could be the asynchronous update buffering technique, which decouples the learning algorithm from the simulation environment. By slightly delaying the update calculations of the learning algorithm into the time span when the simulator does the calculations for the next time step, we can remove the need to explicitly synchronize both tasks (for details, see our previous paper [3]). While this allows for a large performance improvement by removing waiting periods, it is based on the assumption that simulation steps are steady and both tasks do not get delayed unexpectedly. If this assumption is violated by a background task or something similar, and the robot following the leader did some corrections at the very moment, its new position might be even worse than before.

For our goal of learning end-to-end formation control, the precise error bound is less important than the fact that the follower is able to reach a phase where it can continue its episode indefinitely. This usually happens after $60\,000$ steps and works reliable after $100\,000$ steps. Combined with the comparable results from the experiment with the second position of the camera, this shows the flexibility of our approach.

The next logical step is to extend this approach to different formations. We did some initial experiments on varying the follower's angle which show similar results. But modifying the followers angle also required a lot of tuning to the reward and reset function. Therefore the results cannot be compared to those detailed throughout this section and need thorough investigation on generalizability and quantitative performance.

V. Conclusion

Our results show that applying Deep Q-learning to the problem of formation control successfully deals with large input state-spaces. It also removes the tedious work of creating a meaningful representation by hand and is able to adapt to different settings in a flexible way.

References

[1] M. Arai, Y. Sato, R. Suzuki, Y. Kobayashi, Y. Kuno, S. Miyazawa, M. Fukushima, K. Yamazaki, and A. Yamazaki, "Robotic wheelchair moving with multiple companions," in *The 23rd IEEE International Symposium on Robot and Human Interactive Communication*, 2014, pp. 513–518.

[2] J. Saez-Pons, L. Alboul, J. Penders, and L. Nomdedeu, "Multi-robot team formation control in the GUARDIANS project," *Industrial Robot: An International Journal*, vol. 37, no. 4, pp. 372–383, 2010.

[3] M. Knopp, C. Aykın, J. Feldmaier, and H. Shen, "Formation Control using GQ(λ) Reinforcement Learning," in *26th IEEE Int. Symp. Robot and Human Interactive Communication (RO-MAN 2017)*, Lisbon, Portugal, Aug. 2017, pp. 1043–1048.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *NIPS Deep Learning Workshop*, 2013, arXiv: 1312.5602 [cs.LG].

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems 25*, pp. 1097–1105, 2012.

[8] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: a convolutional neural-network approach," *IEEE Trans. Neural Networks*, vol. 8, no. 1, pp. 98–113, 1997.

[9] A. Graves and J. Schmidhuber, "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks," *Advances in Neural Information Processing Systems 21*, pp. 545–552, 2009.

[10] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition," *IEEE Trans. Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 30–42, 2012.

[11] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP 2013)*, May 2013, pp. 6645–6649.

[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *4th Int. Conf. Learning Representations (ICLR 2016)*, 2016, arXiv:1509.02971 [cs.LG].

[13] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli, "The e-puck, a robot designed for education in engineering," in *Proc. 9th Conf. Autonomous Robot Systems and Competitions (Robotica 2009)*, Castelo Branco, Portugal, May 2009, pp. 59–65.

[14] E. Rohmer, S. P. N. Singh, and M. Freese, "V-REP: a versatile and scalable robot simulation framework," in *Proc. IEEE Int. Conf. Intelligent Robots and Systems (IROS 2013)*, Tokyo, Japan, Nov. 2013, pp. 1321–1326.