Deep Reinforcement Learning Methods for Navigational Aids

Bijan Fakhri[1], Aaron Keech[3], Joel Schlosser[3], Ethan Brooks[3], Hemanth Venkateswara[1](✉), Sethuraman Panchanathan[1], and Zsolt Kira[2,3]

[1] School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ 85281, USA
hkdv1@asu.edu

[2] School of Interactive Computing, Georgia Tech, 85 5th St. NW, Atlanta, GA, USA

[3] Georgia Tech Research Institute, 250 15th St. NW, Atlanta, GA, USA

Abstract. Navigation is one of the most complex daily activities we engage in. Partly due to its complexity, navigational abilities are vulnerable to many conditions including Topographical Agnosia, Alzheimer's Disease, and vision impairments. While navigation using solely vision remains a difficult problem in the field of assistive technology, emerging methods in Deep Reinforcement Learning and Computer Vision show promise in producing vision-based navigational aids for those with navigation impairments. To this effect, we introduce GraphMem, a Neural Computing approach to navigation tasks and compare it to several state of the art Neural Computing methods in a one-shot, 3D, first-person maze solving task. Comparing GraphMem to current methods in navigation tasks unveils insights into navigation and represents a first step towards employing these emerging techniques in navigational assistive technology.

Keywords: Navigation · Assistive technology
Reinforcement learning · Topographical agnosia

1 Introduction

From navigating the rooms and hallways of one's own home to navigating a large city, the cognitive functions involved in negotiating an environment to arrive at a predetermined destination are delicate, complex, and in many ways innate. Specialized components of the brain (head direction cells, place cells, grid cells, and border cells) have been shown to be integral to navigation [25]. Although the ability to navigate endows people with independence and self determination, many circumstances can lead to complications in navigation, and a surprising number of people experience such complications. The World Health Organization estimates that 253 million people worldwide have a vision impairment [4] and 21 to 25 million people have Alzheimer's Disease worldwide [7], both of which are known to cause navigation issues [24] among many other conditions.

Navigation itself is a complicated processes, requiring multisensory integration over time and space and a strong dependence on memory. Researchers have determined that efficiently storing and recalling the relationship of landmarks in space is essential to spacial cognition, and thus navigation [24]. There also exists large individual differences in navigation performance: an individual's acuity towards environmental cues, computational mechanisms, and spatial representations that are involved in navigation all are responsible for their navigational aptitute [30]. With so many factors affecting navigational ability, there exists real demand for assistive technology in the space of navigational aids. While the advent of ubiquitous GPS has already benefited many with navigational impairments, small scale and indoor navigation remains a challenge. There does though exist promise in the application of emerging computer vision based technologies for navigational aids.

Deep Learning (DL) and Convolutional Neural Networks (CNNs) have recently emerged to solve complex vision-based tasks [16,18]. Coupled with reinforcement learning, these methods have been shown to learn increasingly complex behavior solely from images [3,22,23,29], from playing Atari games to continuous control. This begs the question: can deep reinforcement learning techniques be employed in assistive technology to aid in navigation? In this paper, we survey the state of the art in Deep Reinforcement Learning methods suited for the high complexity of visual navigation, and present our own technique, GraphMem, designed for such tasks. We also compare the performance of these methods with our own first-person, vision based navigation task built on the ViZDoom 3D research platform [17], shown in Fig. 1. Our results provide insight into how assistive devices can benefit from emerging methods in Artificial Intelligence and provide a platform for future integration into assistive technology.

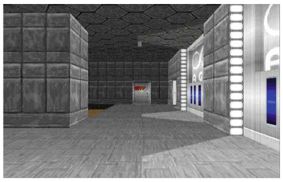

Fig. 1. Agent's point of view in ViZDoom.

2 Related Works

While Deep Q-Networks, Policy Gradients, and related deep reinforcement learning (DRL) methods [3,21] have achieved super-human performance in many previously difficult domains, some seemingly simple tasks have remained out of reach. Specifically, problems with long-term temporal dependencies have proven difficult [28]. Navigation is one such problem; for example, in a first-person maze solving task, it is essential to recognize where you have already been in order to effectively trim the search space.

Several recent papers have validated the ability of deep networks to make sense of 3D environments using visual information, specifically with a focus on navigation tasks. Supervised methods have been developed such as [11], where authors trained a network to infer space through which a robot may travel unobstructed, in order to generate a trajectory for navigating the environment. While there has been success with supervized methods, reinforcement learning paradigms are of predominant interest to our goal, because agent-environment interaction is integral to navigation. Such approaches can be found in the work of [31], where the authors used a double-Q network (D3QN) to achieve obstacle avoidance and path planning in a reinforcement learning setting. DeepMind also showed in "Learning to Navigate in Cities Without a Map" [19] how natural images can be tamed with CNNs paired with LSTMs in vision-based navigation problems. Researchers in [32] also explored transfer between navigation tasks, training the model to navigate one environment and subsequently transferring its learning to a new environment in which the walls and objective have been modified. This work is similar to RL^2 [6], a model which achieves a sizable performance increase of 25.5% between the first and second attempts at the same maze. Our task is similar, but with the added complexity of random start positions between the first and second attempt at a maze. While RL^2 was able to store information in its hidden state, it did not make use of addressable external memory. Due to the complexity of spacial navigation tasks in terms of relational connectivity, we chose to explore methods with the capacity for more complex computation: Memory Augmented Neural Networks (MANNs). DeepMind's work in "Learning to Navigate in Complex Environments" [20] used a stacked LSTM model to solve randomized mazes. While the authors do not employ MANNs in their tests, they stress their applicability to problems of this complexity.

MANNs, sometimes termed "Neural Computers", are characterized by models utilizing an external and addressable memory space [8,9]. This allows them to store and recall information relevant to solving problems that require integrating and processing information over time and space more effectively than standard recurrent networks. For this reason MANNs trained in a Reinforcement Learning setting will be the focus of this work. Specifically, we selected the Differential Neural Computer (DNC) [9] and TARDIS [10] as MANNs to compare to our model, GraphMem. As a baseline, we also compare to a standard feed-forward multilayered perceptron (MLP) and an LSTM [13] based model. There has been work on MANNs used in navigation problems: in [26], authors used a MANN, similar to a Neural Turing Machine [8]. The model was tested in a Minecraft-style maze with discrete movement [15]. Authors also emphasized the use of memory in reinforcement learning tasks in [12], demonstrating the ability of Neural Computers to learn memory-based control tasks. Of these, the most pertinent to the task discussed in this paper is the "water maze" task, in which the agent must first find a hidden objective through random exploration and then subsequently find it again, taking advantage of memories from the initial exploration. Taking inspiration from graph-based representations [2,27], we propose a MANN with graph-like external memory, with the intuition that the spacial connectedness of 3D environments lends itself to a graph-like representation, hence GraphMem.

3 Proposed Method

3.1 GraphMem

In this section we introduce GraphMem, a Memory Augmented Neural Network (MANN) with a novel graph-like external memory (Fig. 2). The choice of a graph structure for the external memory was inspired by the notion that the strong spacial connectivity of 3D environments would be best represented in memory with strong connectivity. Like most MANNs, GraphMem takes in an observation $\mathbf{x}_t$ at time t from the environment and outputs a distribution on actions $\mathbf{a}_t$ to take at time step $t+1$. The magnitude of the ith element $\mathbf{a}_t[i]$ corresponds to the model's confidence in that action relative to all other actions. Observations are transformed into action probabilities by feeding the observation into a CNN, producing state representation vector $\phi_t \leftarrow \text{CNN}(\mathbf{x}_t)$. The representation is fed through the Memory Module generating a context vector $\mathbf{c}_t \leftarrow \text{MM}(\phi_t)$. The context vector represents information read from the memory that is relevant to the current observation. The context vector and state representation are then both fed to the policy (a fully connect neural network), which outputs action probabilities $\mathbf{a}_t \leftarrow \pi(\mathbf{c}_t, \phi_t)$. When the state representation ϕ_t passes through the Memory Module, the module reads from and writes to the memory, determining what to store from ϕ_t and where to store it. Information can thus be stored to be recalled when necessary. This process is outlined below.

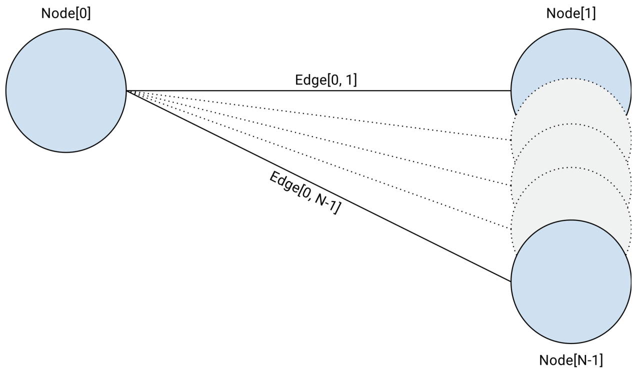

Fig. 2. GraphMem's memory consists of nodes and edges. Information can both be stored in nodes and edges, encouraging relational reasoning.

GraphMem extends the memory structure of the Neural Turing Machine and its successors [8–10] by modelling external memory as a fully connected graph, illustrated in Fig. 2, instead of a sequential array. In practice, the memory graph consists of two arrays, one containing the node data and the other containing the edge data. Figure 3 illustrates the substructures of the memory graph. The node array $\mathbf{N}_{\text{ode}} \in \mathbb{R}^{N\times(A+W)}$ is of size $N \times (A+W)$ where N is the number of nodes, A is the address field size, and W is the word size. The edge array $\mathbf{E}_{\text{dge}} \in \mathbb{R}^{N^2\times W}$ is of size $N^2 \times W$, each edge connects a distinct pairs of nodes. The node array's address field is initialized with unique, sparse random vectors. The content field is initialized with zeros, as well as the edge array. At each time step, GraphMem writes to a single graph node and a single graph edge. The node it writes to is based on a content addressing scheme based on content, while the edge it writes to must be the edge connecting the node written to during the previous time step and the node being written to at the current time step. For example, if GraphMem writes to $\mathbf{N}_{\text{ode}}[i]$ at time t and $\mathbf{N}_{\text{ode}}[j]$ at time $t+1$, the edge it writes to at time $t+1$ is $\mathbf{E}_{\text{dge}}[i,j]$. The discrete and graph-like addressing forces GraphMem to discritize its observations and encourages the network to store information relating observations made in close proximity in both time and space in the edges, an ability indispensable to modeling 3D environments.

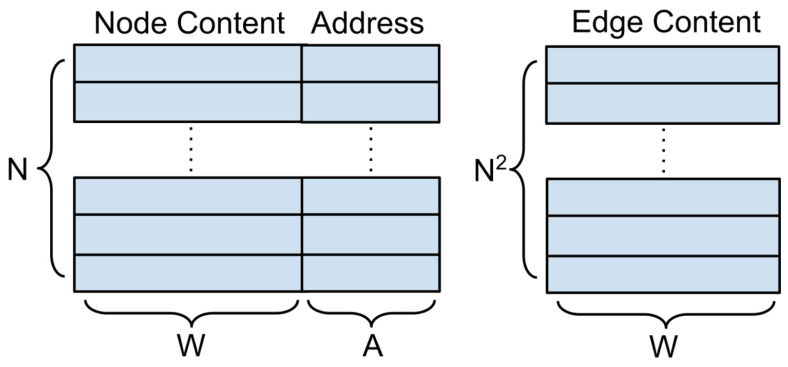

Fig. 3. Memory architecture: node data array (left), edge data array (right).

Similar to TARDIS [10], we use the Gumbel Softmax reparameterization trick [14] for discrete memory addressing to retain the ability to differentiate end-to-end. During memory reads and writes, the state representation vector ϕ_t passes through the Memory Module read/write heads (LSTMs) resulting in address logits vector $\mathbf{w}_t \leftarrow \text{RW}(\phi_t)$. This vector describes the categorical probabilities of reading from or writing to a specific node. Equations 1 and 2 describe how the address logits vector $\mathbf{w}_t$ is transformed into a one-hot vector $\mathbf{m}_t \in \mathbb{R}^N$ describing the memory address of the node to read or write from.

$$\mathbf{g}_t \leftarrow gumbel(\mathbf{w}_t) \tag{1}$$

$$\mathbf{m}_t = (one\_hot(argmax(\mathbf{g}_t)) - \mathbf{g}_t) + \mathbf{g}_t \tag{2}$$

Equation 2 features an *argmax* and *one_hot* operation, which are not differentiable. To circumvent this, the gradient only flows through $\mathbf{g}_t$, the last term, bypassing $(one\_hot(argmax(\mathbf{g}_t)) - \mathbf{g}_t)$. This estimates the derivative while allowing backpropagation through a discrete addressing mechanism. Details of the Gumbel Softmax function are described in [14].

3.2 Maze Task

The ViZDoom maze task was designed to reveal how effectively an agent can re-navigate to a location it has been to before, having started at a new location. Figure 4 shows a bird's eye view of the map and screenshots of the agent's point of view. Notice, the maze is not "simply connected" as it features detached walls that can fool more simple maze solving algorithms. The goal of this task is to find the "health pack" hidden in the maze. Each episode consists of two phases. For each phase the agent spawns in a random room and must search the maze for a "health pack". The agent is rewarded, on a per-episode basis, proportionally to the number of steps it takes to reach the goal ("health pack"). The fewer total steps taken (phase1 + phase2), the higher the agent's reward. In both phases of an episode the agent is given the same maze, so that the agent can make use of what was learned about the maze in Phase 1 when looking for the "health pack" in Phase 2. It is important to note that the agent is rewarded in proportion to the summation of steps taken in each phase. The agent will thus learn to minimize the total number of steps and in no way is directed to use its memory to optimize the second encounter. At the conclusion of an episode, both the locations of the "health pack" and furniture in the rooms is randomized, so that the agent must learn a policy that memorizes the maze's composition using its external memory only. This is to prevent the agent from memorizing the maze using the parameters of the model, which is slow and poorly replicates a real navigation scenario.

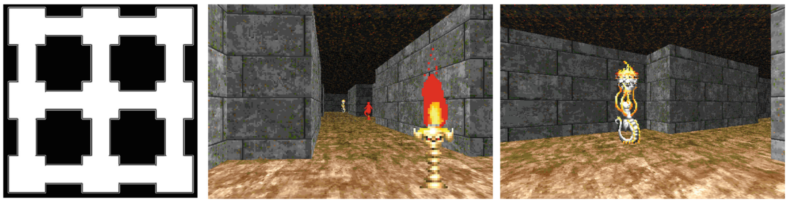

Fig. 4. Blueprint of the maze (left) Screenshots of the rooms (middle, right).

The maze environment consists of 9 rooms connected by hallways (shown in Fig. 4). All of the walls are identical. The only unique features in the rooms are pieces of "furniture" placed in the rooms, one piece of furniture per room. The goal is also placed in a random room at a random offset from the center of the room. This makes seeing the goal from across the maze non-trivial. Because the hallways are narrower than the rooms, furniture and the goal are not necessarily visible from another room. The agent may also get "caught" on the walls of the room, so the agent must learn efficient movement as well as an efficient exploration policy to maximize its reward. The maze was designed to be non-simply connected, meaning agents that cannot identify and address loop closures may loop indefinitely.

3.3 Training

We trained all of the models using the Asynchronous Advantage Actor-Critic (A3C) algorithm [22], which allows for training a model using many distinct instances of the environment in parallel. Parameter updates from the distinct instances are applied asynchronously to a master copy of the policy, which is periodically copied down to the worker copies of the policy that are interacting with the environment. The gradient is describe in Eq. 3, with policy π, return R_t, value function V, and model parameters θ. The model entropy $H(\pi(\mathbf{x}_t;\theta'))$ is also considered in the gradient to discourage premature convergence to sub-optimal policies (scaled by hyperparameter $\beta = 10^{-4}$).

$$\nabla_{\theta'} \log \pi(\mathbf{a}_t|\mathbf{x}_t;\theta')(R_t - V(\mathbf{x}_t;\theta_v)) + \beta \nabla_{\theta'} H(\pi(\mathbf{x}_t;\theta')) \tag{3}$$

The models were trained on a 12-core Xeon machine with an Nvidia GTX 1080ti using TensorFlow 1.3.0 [1]. Each model was trained for 30 million time steps (∼12 h). Figure 5 shows the training graphs for all models. It is interesting to note that all models show meager performance until 10–15M time steps of training. For our tests, we used the DeepMind implementation of the Differential Neural Computer [9]. The LSTM and MLP models used were public A3C [22] implementations proven to work on OpenAI Gym benchmark suite [5] environments. We used our own implementation of TARDIS as a public version was not available at the time of writing.

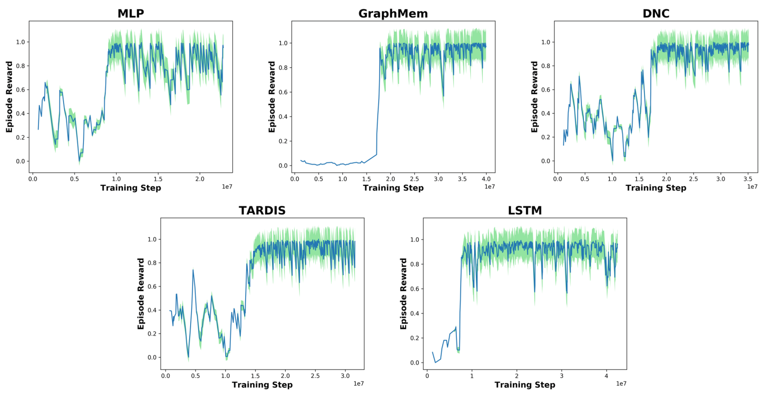

Fig. 5. Training graphs for all models.

4 Results

After training, all models were subjected to 123456 episodes of testing. During testing, the parameters of the network were frozen by disabling backpropagation and the models were subjected to the maze environment for evaluation. Figure 6 illustrates the average number of steps taken by the models in solving the maze tasks as well as the percentage improvement of steps between Phase 1 and Phase 2 of an episode. TARDIS and DNC proved to be the fastest models, while GraphMem was the slowest and the MLP and LSTM remain in middle of the pack. With regards to leveraging memory, GraphMem saw the greatest percentage improvement (percentage difference in number of steps between Phase 1 and Phase 2) of all the models, followed by LSTM. It is surprising to note that the two other MANNs were unable to capitalize on having already seen the maze, both models performed about as well as the memoryless MLP model (Table 1).

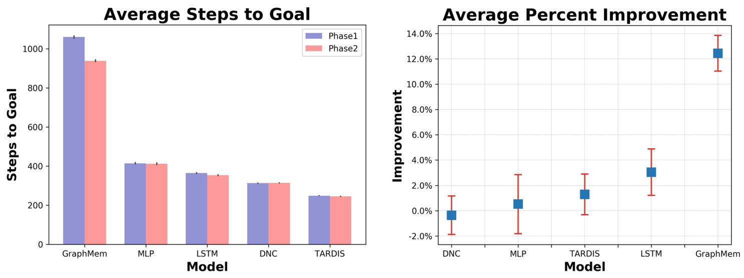

Fig. 6. Average steps to goal (left) percent improvement from Phase 1 to Phase 2 (right) both with 95% confidence intervals.

Table 1. Results with 95% confidence intervals.

	Phase1 $\bar{\mu}_{\text{steps}}$	Phase2 $\bar{\mu}_{\text{steps}}$	Steps sum	Improvement
TARDIS	248.92 ± 2.82	245.67 ± 2.88	494.59 ± 4.03	$1.29 \pm 1.61\%$
DNC	312.97 ± 3.39	314.06 ± 3.34	627.03 ± 4.75	$-0.36 \pm 1.52\%$
MLP	414.54 ± 6.57	412.27 ± 6.68	826.81 ± 9.37	$0.52 \pm 2.33\%$
LSTM	364.74 ± 4.85	353.56 ± 4.74	718.31 ± 6.78	$3.05 \pm 1.83\%$
Ours	1060.24 ± 8.43	938.76 ± 7.95	1999.00 ± 11.59	$11.45 \pm 1.41\%$

5 Conclusion and Future Work

Vision-based navigation remains a difficult problem in assistive technology. While Deep Reinforcement Learning methods for navigation are still in their infancy, this work highlights some promising approaches towards that goal. Improved methods will be beneficial in compensating for navigational difficulties faced by people with disabilities. Future work will emphasize memory access and content analysis - with the goal of human interpretable memory contents. We also intend to explore integrating this work or similar models with wearable computers, in order to provide real-time navigational assistance to users that are blind or have other vision impairments.

References

1. Abadi, et al.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems (2016)
2. Allamanis, M., Brockschmidt, M., Khademi, M.: Learning to Represent Programs with Graphs. In: ICLR (2018)
3. Bengio, Y.: Continuous control with deep reinforcement learning. Foundations and Trends®. Mach. Learn. **2**(1), 1–127 (2009)
4. Bourne, R.R.A., et al.: Magnitude, temporal trends, and projections of the global prevalence of blindness and distance and near vision impairment: a systematic review and meta-analysis. Lancet Global Health **5**(9), e888–e897 (2017)
5. Brockman, G., et al.: OpenAI Gym (2016)
6. Duan, Y., Schulman, J., Chen, X., Bartlett, P., Sutskever, I., Abbeel, P.: RL^2: fast reinforcement learning via slow reinforcement learning. arXiv, pp. 1–14 (2016)
7. Duthey, B.: Background Paper 6.11 Alzheimer Disease and other Dementias, Update on 2004. World Health Organization, pp. 1–77, February 2013
8. Graves, A., Wayne, G., Danihelka, I.: Neural Turing Machines. arXiv, pp. 1–26, October 2014
9. Graves, A., et al.: Hybrid computing using a neural network with dynamic external memory. Nat. Res. (2016)
10. Gulcehre, C., Chandar, S., Bengio, Y.: Memory Augmented Neural Networks with Wormhole Connections. arXiv, pp. 1–27 (2017)
11. Gupta, S., Davidson, J., Levine, S., Sukthankar, R., Malik, J.: Cognitive mapping and planning for visual navigation. In: CVPR (2017)
12. Heess, N., Hunt, J.J., Lillicrap, T.P., Silver, D.: Memory-based control with recurrent neural networks (2015)
13. Hochreiter, S., Urgen Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
14. Jang, E., Gu, S., Poole, B.: Categorical Reparameterization with Gumbel-Softmax. In: International Conference on Learning Representations, pp. 1–13 (2017)
15. Johnson, M., Hofmann, K., Hutton, T., Bignell, D.: The Malmo platform for artificial intelligence experimentation. In: IJCAI International Joint Conference on Artificial Intelligence, January 2016, pp. 4246–4247 (2016)
16. Karpathy, A., Li, F.F.: Deep visual-semantic alignments for generating image descriptions. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 39, no. 4, pp. 3128–3137 (2015)

17. Kempka, M., Wydmuch, M., Runc, G., Toczek, J., Jaskowski, W.: ViZDoom: a doom-based AI research platform for visual reinforcement learning. In: IEEE Conference on Computational Intelligence and Games, CIG (2017)
18. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances In Neural Information Processing Systems, pp. 1–9 (2012)
19. Mirowski, et al.: Learning to Navigate in Cities Without a Map (2018)
20. Mirowski, P., et al.: Learning to navigate in complex environments. ICLR (2017)
21. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
22. Mnih, et al.: Asynchronous methods for deep reinforcement learning. arXiv preprint, vol. 48, pp. 1–28. arXiv:1602.01783v1 [cs.LG] (2016)
23. Mnih, V., et al.: Playing Atari with deep reinforcement learning. arXiv (2013). https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf
24. Monacelli, A.M., Cushman, L.A., Kavcic, V., Duffy, C.J.: Spatial disorientation in Alzheimer's disease: the remembrance of things passed. Neurology **61**(11), 1491–1497 (2003)
25. Moser, M.B., Rowland, D.C., Moser, E.I.: Place cells, grid cells, and memory. Cold Spring Harb. Perspect. Biol. **7**(2), a021808 (2015)
26. Oh, J., Chockalingam, V., Singh, S., Lee, H.: Control of Memory, Active Perception, and Action in Minecraft. arXiv:1605.09128 [cs] (2016)
27. Sanchez-Gonzalez, A., et al.: Graph networks as learnable physics engines for inference and control (2018)
28. Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., Lillicrap, T.: One-shot Learning with Memory-Augmented Neural Networks. arXiv:1605.06065 [cs], May 2016
29. Schulman, J., Levine, S., Jordan, M., Abbeel, P.: Trust region policy optimization. In: ICML 2015, p. 16, February 2015
30. Wolbers, T., Hegarty, M.: What determines our navigational abilities? Trends Cogn. Sci. **14**(3), 138–146 (2010)
31. Xie, L., Wang, S., Markham, A., Trigoni, N.: Towards Monocular Vision based Obstacle Avoidance Through Deep Reinforcement Learning. Robotics: Science and Systems Workshop 2017: New Frontiers for Deep Learning in Robotics (2017)
32. Zhang, J., Springenberg, J.T., Boedecker, J., Burgard, W.: Deep reinforcement learning with successor features for navigation across similar environments. In: IEEE International Conference on Intelligent Robots and Systems, September 2017, pp. 2371–2378 (2017)