Visual Navigation in Real-World Indoor Environments Using End-to-End Deep Reinforcement Learning

Jonáš Kulhánek[1], Erik Derner[2], and Robert Babuška[3]

***Abstract*—Visual navigation is essential for many applications in robotics, from manipulation, through mobile robotics to automated driving. Deep reinforcement learning (DRL) provides an elegant map-free approach integrating image processing, localization, and planning in one module, which can be trained and therefore optimized for a given environment. However, to date, DRL-based visual navigation was validated exclusively in simulation, where the simulator provides information that is not available in the real world, e.g., the robot's position or segmentation masks. This precludes the use of the learned policy on a real robot. Therefore, we present a novel approach that enables a direct deployment of the trained policy on real robots. We have designed a new powerful simulator capable of domain randomization. To facilitate the training, we propose visual auxiliary tasks and a tailored reward scheme. The policy is fine-tuned on images collected from real-world environments. We have evaluated the method on a mobile robot in a real office environment. The training took approximately 30 hours on a single GPU. In 30 navigation experiments, the robot reached a 0.3-meter neighbourhood of the goal in more than 86.7 % of cases. This result makes the proposed method directly applicable to tasks like mobile manipulation.**

***Index Terms*—Vision-based navigation, reinforcement learning, deep learning methods.**

I. Introduction

VISION-BASED navigation is essential for a broad range of robotic applications, from industrial and service robotics to automated driving. The wide-spread use of this technique will be further stimulated by the availability of low-cost cameras and high-performance computing hardware.

Conventional vision-based navigation methods usually build a map of the environment and then use planning to reach the goal. They often rely on precise, high-quality stereo cameras and additional sensors, such as laser rangefinders, and are computationally demanding. As an alternative, end-to-end deep-learning systems can be used that do not employ any map. They integrate image processing, localization, and planning in one module or agent, which can be trained and therefore optimized for a given environment. While the training is computationally demanding, the execution of the eventual agent's policy is computationally cheap and can be executed in real time (sampling times around 50 ms), even on light-weight embedded hardware, such as NVIDIA Jetson. These methods also do not require any expensive cameras.

The successes of deep reinforcement learning (DRL) on game domains [1]–[3] inspired the use of DRL in visual navigation. As current DRL methods require many training samples, it is impossible to train the agent directly in a real-world environment. Instead, a simulator provides the training data [4]–[14], and domain randomization [15] is used to cope with the simulation-to-reality gap. However, there are several unsolved problems associated with the above simulator-based methods:

- The simulator often provides the agent with features that are not available in the real world: the segmentation masks [4], [6], [10], distance to the goal, stopping signal [4]–[7], [11], [12], [14], etc. This information is given either as one of the agent's inputs [4], [10] or in the form of an auxiliary task [6]. While this improves the learning performance, it precludes a straightforward transfer from simulation-based training to real-world deployment. For auxiliary tasks using segmentation masks during training [6], another deep neural network could be used to annotate the inputs [16], [17]. However, this would introduce additional overhead and noise to the process, and it would diminish the performance gain.
- Another major problem with the current approaches [4]–[7], [11], [12], [14] is that during the evaluation, the agent uses yet other forms of input provided by the simulator. In particular, it relies on the simulator to terminate the navigation as soon as the agent gets close to the goal. Without this signal from the simulator, the agent never stops, and after reaching the goal, it continues exploring the environment. If we provide the agent with the termination signal during training, in some cases, the agent learns an efficient random policy, ignores the navigation goal, and only explores the environment efficiently.

To address these issues, we propose a novel method for DRL visual navigation in the real world, with the following contributions:

1) We developed a fast and realistic environment simulator

Manuscript received: October 15, 2020; Revised February 16, 2021; Accepted February 16, 2021.

This paper was recommended for publication by Editor Eric Marchand upon evaluation of the Associate Editor and Reviewers' comments.

This work was supported by the European Regional Development Fund under the project Robotics for Industry 4.0 (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000470). Robert Babuška was supported by the European Union's H2020 project Open Deep Learning Toolkit for Robotics (OpenDR) under grant agreement No 871449. A part of the computational resources was supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

[1]Jonáš Kulhánek is with the Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, 16636 Prague, Czech Republic jonas.kulhanek@cvut.cz

[2]Erik Derner is with the Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, 16636 Prague, Czech Republic and with the Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, 16627 Prague, Czech Republic erik.derner@cvut.cz

[3]Robert Babuška is with Cognitive Robotics, Faculty of 3mE, Delft University of Technology, 2628 CD Delft, The Netherlands and with the Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, 16636 Prague, Czech Republic r.babuska@tudelft.nl

Digital Object Identifier (DOI): see top of this page.

based on Quake III Arena [18] and DeepMind Lab [19]. It allows for pre-training the agents quickly, and thanks to the high degree of variation in the simulated environments, it helps to bridge the reality gap. Furthermore, we propose a procedure to fine-tune the pre-trained agent on a dataset consisting of images collected from the real-world environment. This enables us to use a lot of synthetic data collected from the simulator and still train on data visually similar to the target real-world environment.

2) We designed a set of auxiliary tasks that do not require any input that is not readily available to the robot in the real world. Therefore, we can use the auxiliary tasks both during the pre-training on the simulator and during the fine-tuning on previously unseen images collected from the real-world environment.
3) Similarly to [8], [9], we use an improved training procedure that forces the agent to detect whether it reached the goal automatically. This enables the direct use of the trained policy in the real world, where the information from the simulator is not available.
4) To demonstrate the viability of our approach, we report a set of experiments in a real-world office environment, where the agent was trained to find its goal given by an image.

II. Related work

The application of DRL to the visual navigation domain has recently attracted a lot of attention [4]–[14]. This is thanks to the deep learning successes in the computer vision [17], [20] and gaming domains [2], [3], [21]. However, most of the research was done in simulated environments [4]–[10], [13], [14], where one can sample an almost arbitrary number of trajectories. The early methods used a single fixed goal as the navigation target [7]. More recent approaches are able to separate the goal from the navigation task and pass the goal as an input either in the form of an embedding [4], [9], [14] or as an input image [5], [6], [11], [12]. Visual navigation was also combined with natural language in the form of instructions for the agent [10].

The use of auxiliary tasks to stabilize the training was proposed in [7] and later applied to visual navigation [6]. In our work, we use a similar set of auxiliary tasks, but instead of using segmentation masks to build the internal representation of the observation, we use the camera images directly. This allows us to apply our method to the real world without labelling the real-world images.

Most of the current approaches [4]–[14] were validated only in simulated environments, where the simulator provides the agent with signals that are normally not available in the real-world setting, such as the distance to the goal or segmentation masks. This simplifies the problem, but at the same time, precludes the use of the learnt policy on a real robot.

There are methods [11], [12], [14] which attempt to apply end-to-end DRL to real-world domains. In [14], the authors trained their agent to navigate in Google Street View, which is much more photo-realistic than other simulators. They, however, did not evaluate their agent on a real robot. In [11], a mobile robot was evaluated in an environment discretized into a grid with about 27 states, which is the order of magnitude lower than our method. Moreover, they did not use visual features directly but used ResNet [20] features instead. This makes the problem easier to learn, but the final environment has to have a lot of visual features recognizable by the trained ResNet. In our case, we do not need such a requirement, and our agent is much more general.

Generalization across environments is discussed in [12]. The authors trained the agent in domain-randomized maze-like environments and experimented with a robot in a small maze. Their agent, however, does not include any stopping action, and the evaluator is responsible for terminating the navigation. Our evaluation is performed in a realistic real-world environment, and we do not require any external signal to stop the agent.

We focus on end-to-end deep reinforcement learning. We believe that it has the potential to overcome the limitations of and have superior performance to other methods which use deep learning or DRL as a part of their pipeline, e.g., [22], [23]. We also do not consider pure obstacle avoidance methods such as [24]–[26], or methods relying on other types of input apart from the camera images, e.g., [27], [28].

III. Method

We modify and extend our method [6] to adapt it to real-world environments. We design a powerful environment simulator that uses synthetic scenes to pre-train the agent. The agent is then fine-tuned on real-world images. The implementation is publicly available on GitHub[1], including the source code of the simulator[2].

A. Network architecture & training

The architecture from our prior work [6] uses a single deep neural network as the learning agent. It outputs a probability distribution over a discrete set of allowed actions. The input to the agent is the visual output of the camera mounted on the robot, and an image of the goal, i.e., an image taken from the agent's target pose. The previous action and reward are also used as inputs to the agent.

The network is trained using the Parallel Advantage Actor-Critic (PAAC) algorithm [29] with off-policy critic updates. We have also considered the Proximal Policy Optimization (PPO) algorithm [30] as an alternative. While the used network architecture can, in principle, accommodate various training algorithms, we have chosen the PAAC algorithm thanks to the performance it demonstrated in our previous work [6].

The network uses a stack of convolutional layers at the bottom, Long Short-Term Memory (LSTM) [31] in the middle, and actor and critic heads. To improve the training performance, the following auxiliary tasks were used in [6]: pixel control, reward prediction, reconstruction of the depth image, and of the observation and target image segmentation masks.

[1]https://github.com/jkulhanek/robot-visual-navigation
[2]https://github.com/jkulhanek/dmhouse

Each auxiliary task has its own head, and the overall loss function is computed as a weighted sum of the individual losses. For further details, refer to [6], [7].

To be able to train the agent on real-world images, we modified the set of auxiliary tasks. We no longer use the segmentation masks as they cannot be obtained from the environment without manual labelling. We have therefore replaced the two segmentation auxiliary tasks with two new auxiliary tasks: one to reconstruct the observation image and the other one to reconstruct the target image, see Fig. 1. This guides the convolutional base part of the network to build useful features using unsupervised learning. We hypothesize that having the auxiliary tasks share the latent space with the actor and the critic will have a positive effect on the training performance. This hypothesis is verified empirically in Section IV.

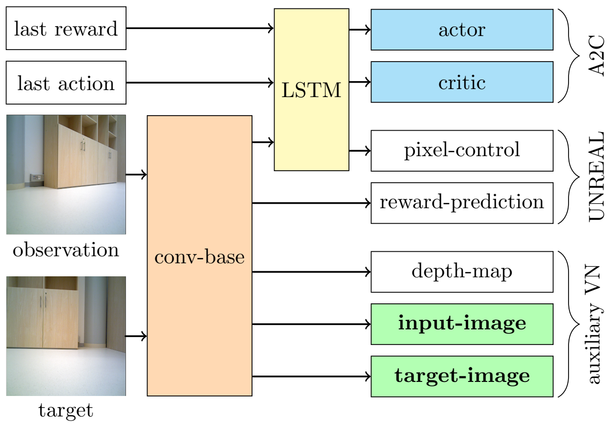

Fig. 1. Neural network architecture. It is similar to [6] with the difference in the visual navigation (VN) auxiliary tasks: we use the raw camera images instead of the segmentation masks, which are not readily available in the real-world environment.

Similarly to [9], we have also modified the action set to include a new action called *terminate*. This action stops the navigation episode and, therefore, enables the trained agent to navigate in a real-world environment using only the camera images, without additional sensors providing its actual pose in the environment. During training, when the episode terminates with the *terminate* action, the agent receives either a positive or a negative reward based on its distance to the goal. The episode can also terminate (with a negative reward) after a predefined maximum number of steps.

B. Environment simulator

Since DRL requires many training samples, we first pre-train the agent in a simulated environment and then fine-tune on images collected from the real world. We have designed a novel, fast, and realistic 3D environment simulator that dynamically generates scenes of office rooms. It is based on DeepMind Lab [7], and it uses the Quake III Arena [18] rendering engine. We have extended the simulator with office-like models and textures. Rooms are generated by placing random objects along the walls, e.g., bookshelves, chairs, or cabinets. In total, there are seven object classes in the simulator and up to 14 available spaces to which an object can be placed. When generating a room layout, nine randomly selected spaces are filled with random objects.

For each room layout, a new map is generated, which is then compiled and optimized using tools designed for Quake. To speed up the training, we keep the same room layout for 50 episodes before reshuffling the objects. The simulator also randomizes lighting conditions for each episode. After generating the room layout, the target object is selected from the set of objects placed in the room, and a target image is taken from the proximity of the object. Fig. 2 shows four random examples of images from the environment.

Compared to other simulators, e.g., Habitat [32], or AI2-THOR [33], our simulator is less photo-realistic while being faster and capable of domain randomization. We do not require the simulator to be photo-realistic since it is used only for pre-training. Moreover, when using a feature-rich environment simulator, the agent often learns to use only the recognized features while ignoring the 3D properties of the environment [5]. We claim that by having less detail in the environments, the agent can focus on the navigation using the 3D properties of the environment instead of recognizing the features.

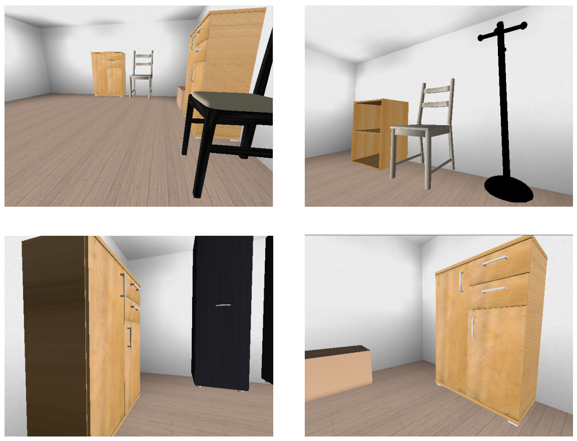

Fig. 2. Images taken from our environment simulator.

C. Fine-tuning on real-world data

Since the simulated environment does not precisely match the real-world environment, we fine-tune the agent on a set of real-world images. Throughout the rest of this paper, we restrict the motion of the robot to a rectangular grid, both for the image collection and for the final evaluation experiments.

To collect the training data, we placed the mobile robot equipped with an RGB-D camera in the real-world environment. We programmed the robot to automatically collect a set of images and depth maps from each point of a rectangular grid in all four cardinal directions. Odometry was used to derive and store the robot's pose (position and orientation) from which the image was taken. In this way, we obtained a dataset consisting of tuples $(x, y, \phi, i, \text{camera image}, \text{depth map})$, where x, y, and ϕ are the pose coordinates, and i is the index of the observation image, as several images were taken from each pose.

The pre-trained agent was then fine-tuned on the collected dataset. It was initialized at a random pose, and its current policy was used to move to the subsequent pose on the grid. For each pose, a random image was sampled from the dataset, and this process repeated until the episode terminated. In this way, the agent was trained in an on-policy fashion on rollouts generated from previously collected data.

D. Real-world deployment

After the training, the agent uses its trained policy and requires only the camera observation and the target image – no depth image or localization is necessary. To deploy our method to a new environment, we propose the following procedure. First, the agent is pre-trained in a simulated environment using our simulator. RGB-D images are then semi-automatically collected from the real-world environment, and the agent is fine-tuned on this dataset. Finally, the agent can be placed in the real-world environment with an RGB camera as its only sensor. Note that the depth maps are not needed for the deployed robot.

IV. Experiments

First, we compared several algorithms on our 3D simulated environment. Then, the pre-trained agent was fine-tuned on the real-world images. We used these real-world images to evaluate the performance of our method. Finally, we evaluated the method on the real robot.

A. Hyperparameter configuration

The input images to the network were downsized to 84×84 pixels. The shared convolutional part of the deep neural network had four convolutional layers. The first convolutional layer had 16 features, kernel size 8×8, and stride 4. The second convolutional layer had 32 features, kernel size 4×4, and stride 2. The third layer had 32 features with kernel size 4×4 and stride 1. The first fully-connected layer had 512 features and the LSTM also had 512 output features and a single layer. The deconvolutional networks used in the auxiliary tasks had two layers with kernel sizes 4×4 and strides 2, first having 16 output features. For image auxiliary tasks, the first layer was shared. The pixel control task used a similar architecture, but it had a single fully-connected layer with 2592 output features at the bottom, and it was duelled, as described in [7], [34]. We used the discount factor $\gamma = 0.99$ for simulated experiments and $\gamma = 0.9$ for the real-world dataset. The use of smaller γ for the real-world data was motivated by shorter episodes to make the algorithm converge faster. The training parameters are summarized in Table I. The learning rate decays with f, which is the total number of frames presented to the learning algorithm. The actor weights, critic weight, etc., correspond to the weights of each term in the total loss [6]. Note that the hyper-parameters of other methods (PPO, PAAC, UNREAL) are omitted from this paper for brevity, but they are included in the published source code.

TABLE I
Our method's parameters.

name	value
discount factor (γ)	0.99 simulated / 0.9 real-world dataset
maximum rollout length	20 steps
number of environment instances	16
replay buffer size	2 000 samples
optimizer	RMSprop
RMSprop alpha	0.99
RMSprop epsilon	10^{-5}
learning rate	$\max(0, 7 \times 10^{-4}(1 - \frac{f}{4 \times 10^7}))$
max. gradient norm	0.5
entropy gradient weight	0.001
actor weight	1.0
critic weight	0.5
off-policy critic weight	1.0
pixel control weight	0.05
reward prediction weight	1.0
depth-map prediction weight	0.1
input image reconstruction weight	0.1
target image reconstruction weight	0.1
pixel control discount factor	0.9
pixel control downsize factor	4
auxiliary VN downsize factor	4

B. Experiment configuration

For both the simulated environment and the real-world dataset, we have evaluated the performance of the proposed algorithm and compared it to the performance of relevant baseline algorithms. The results were computed from 1000 trials, where the starting position and the goal were sampled randomly. We present the mean cumulative reward and the success rate – the percentage of cases when the agent reached the goal.

C. Simulated environment

In the simulated environment, we trained the algorithm for 8×10^6 training frames. We gave the agent a reward of 1 if it reached the goal and stopped using the *terminate* action. We gave it a reward of -0.1 if it used the *terminate* action incorrectly close to (and looking at) an object of a different type than the goal, e.g., a bookcase while the goal was a chair. Otherwise, the reward was 0. In the simulated environment, we did not stop the agent when it used the *terminate* action incorrectly. This leads to improved training efficiency because generating a new room layout is time-consuming.

The evaluation was done in 100 randomly generated environments. A total of 1000 simulations were evaluated with the initial positions and the goals randomly sampled. The success rate, the mean travelled distance, the mean number of simulation steps, and their standard deviations are given in Table II. The mean number of simulation steps was computed from the successful episodes only, i.e., those episodes which resulted in the agent signalling the goal correctly.

The training performance can be seen in Fig. 3. Our algorithm is compared to PAAC [29], which has a comparable performance to A3C [21], and with PPO [30]. We compared the proposed method also with the UNREAL method [7].

TABLE II
METHOD PERFORMANCE ON SIMULATED ENVIRONMENTS.

algorithm	success rate	distance travelled (m)	simulation steps
ours	**1.000**	**7.691±3.415**	**41.552±25.717**
PAAC	0.420	66.913±30.329	398.214±276.642
PPO	0.408	81.122±25.564	457.404±288.726
UNREAL	0.999	8.322±6.187	45.541±50.349

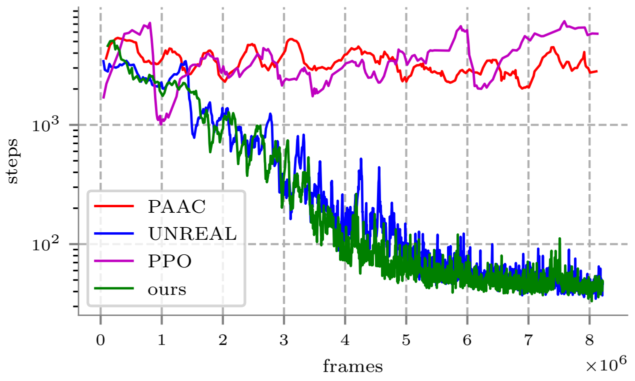

Fig. 3. The plot shows the average episode length during training in the simulated environment. The PAAC, UNREAL, PPO, and our algorithm are compared. Note that we plot the average episode length, which may not correlate with the success rate – the probability of signalling the goal correctly. In the case of UNREAL and our algorithm, however, this metric better shows the asymptotic behaviour of the two algorithms since both converged to the success rate of one.

D. Real-world dataset

In an office room, we used the TurtleBot 2 robot (Fig. 4) to collect a dataset of images taken at grid points with a 0.2 m resolution. Examples of these images can be seen in Fig. 5. When we collected the dataset, we estimated the robot pose through odometry. The odometry was also used for the evaluation of the trained agent in the experimental environment to assess whether the agent fulfilled its task and stopped close to the goal.

The target was sampled from the set of robot positions near a wall or near an object and facing the wall or the object. The

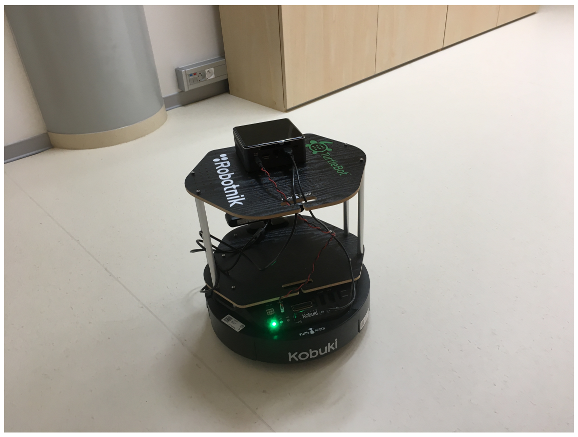

Fig. 4. Mobile robot TurtleBot 2 in the experimental environment.

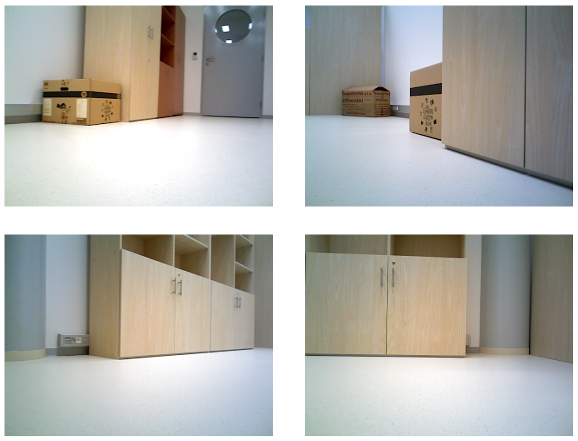

Fig. 5. Training images from the real-world dataset.

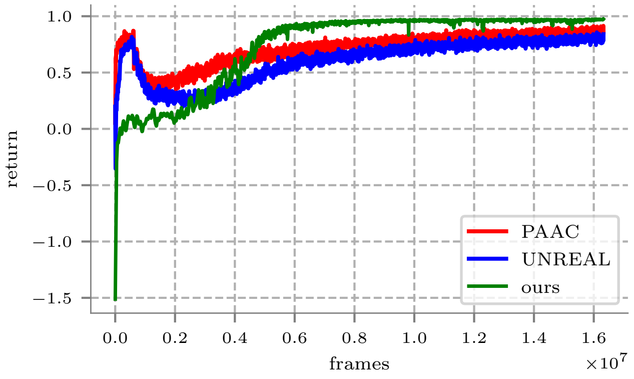

Fig. 6. The average return during training on the simulated environment.

initial position was uniformly sampled from the set of initial positions, and the initial orientation was chosen randomly. The set of initial positions is adapted via curriculum learning [6], [14]. At the beginning of the training, it is defined as all non-target positions with a distance of up to three steps to the goal state. Between frames 0.5×10^6 and 5×10^6, this distance gradually increases to the maximum distance given by the environment.

The final position of the robot was considered correct when its Euclidean distance from the target position was at most 0.3 m, and the difference between the robot orientation and the target orientation was at most 30°. The tolerance was set to compensate for the odometry inaccuracy. For the purpose of training, we had chosen the reward to be equal to 1 when the agent reached and stopped at its target. We penalized the agent with a reward of -0.01 if it tried to move in a direction that would cause a collision. Otherwise, the reward was zero.

The algorithm was trained on 1.65×10^7 frames. The training performance can be seen in Fig. 6. Our method was compared to the PAAC algorithm [29], and the UNREAL algorithm [7], modified in the same way as described in Section IV-C. All agents were pre-trained in the simulated environment. We evaluated the algorithms in a total of 1000 simulations with the initial positions and the goals randomly sampled. The success rate, the mean distance from the goal

(goal distance) when the goal was signalled, and the mean number of simulation steps are given in Table III. We also show a comparison with the same methods trained on the real images from scratch (labels beginning with *np*, which stands for no pre-training). Same as before, the mean number of simulation steps was computed from the successful episodes only. For comparison, we also report the shortest path and the performance of a random agent. The random agent selects random movements, and when it reaches the target, the ground truth information is used to signal the goal.

TABLE III
METHOD PERFORMANCE ON REAL-WORLD DATASET.

algorithm	success rate	goal distance (m)	steps on grid
ours	**0.936**	**0.145±0.130**	**13.489±6.286**
PAAC	0.922	0.157±0.209	14.323±10.141
UNREAL	0.863	0.174±0.173	14.593±9.023
np ours	0.883	0.187±0.258	15.880±7.022
np PAAC	0.860	0.243±0.447	13.699±6.065
np UNREAL	0.832	0.224±0.358	15.676±6.578
random	0.205	1.467±1.109	147.956±88.501
shortest path	–	0.034±0.039	12.595±5.743

np = *not pre-trained*

E. Real-world evaluation

Finally, to evaluate the trained network in the real-world environment, we have randomly chosen 30 pairs of initial and target states. The trained robot was placed in an initial pose, and it was given a target image. The results are summarized in Table IV, where we also include the simulation of the trained agent on the real-world image dataset. We show the mean number of simulation steps, the mean distance from the goal (goal distance), and the success rate, where the mean number of simulation steps was computed from the successful episodes only. The latter results slightly differ from the results reported in the first row of Table III, as in this case, a different smaller set of the initial state – goal state pairs was used.

TABLE IV
COMPARISON OF REAL-WORLD AND SIMULATION EXPERIMENTS.

evaluation	success rate	goal distance (m)	steps on grid
real mobile robot	0.867	0.175±0.101	15.153±6.455
simulated on real images	0.933	0.113±0.109	14.750±6.583

V. DISCUSSION

A. Alternative agent objective

In this work, we framed the navigation problem as the ability of the agent to reach the goal and stop there. To this end, we introduced the *terminate* action and trained the agent to use it. Alternatively, we could stick to the approach of previous visual navigation work [4], [6]–[8], [12] in which the agent is stopped by the simulator. In the real-world environment, a localization method [35], [36] would then have to be used to detect whether the agent reached the goal. However, this would introduce unnecessary computational overhead for the robot, and it would obscure the evaluation, as a navigation failure could be caused by either of the two systems.

B. Simulated environments

Training the agent on the simulated environments was difficult since the agent was supposed to generalize across different room layouts. In this case, the agent did not learn to navigate to a given target by using the shortest path but to locate the target object in the environment first and then to navigate to it while minimizing the total number of steps. From some initial positions, the target could not be seen directly, and the agent had to move around to see the target. Also, sometimes, there were many instances of the same object, complicating the task even more.

Our method achieved the best results of all compared methods, closely followed by the UNREAL algorithm [7], see Table II. This clearly indicates the positive effect of using auxiliary tasks for continuous spaces. The similarity between our method and the UNREAL algorithm is that both were close to the optimal solution and could not improve anymore. This is implied by the fact that both methods reached and signalled the goal successfully almost always. The difference between PPO and PAAC was marginal during the training. Both algorithms were oscillating throughout the training, and unlike our method and UNREAL, they were not able to stabilize and shorten the episode length.

C. Real-world dataset

In the case of the real-world dataset (Table III), the goal was to learn a robust navigation policy in a noisy environment. The noisiness came from the fact that we did not have access to the precise robot positions and orientations when the dataset was collected, but only to their estimates based on odometry. Our method achieved the best performance. It was followed by the raw PAAC algorithm [29]. The UNREAL method [7] was the worst of these methods. We see that for these algorithms, the average number of steps to get to the target is very close to the minimal number of steps.

We did not require the agent to end up precisely in the correct position, but only in the 30 cm neighbourhood of the target position. Within this neighbourhood, the reward was always the same, regardless of the actual distance from the target. The noisiness in the position labels in the dataset might cause the agent to stop closer to the goal than it was necessary to meet a safety margin at the cost of more steps.

D. Ablation study

The results in Table III demonstrate the effectiveness of each of our contributions separately. A substantial performance improvement can be observed when using pre-training regardless of the training algorithm. Although the simulated environment used continuous space, it looked visually different, and the set of control actions was different, the pre-training was still remarkably beneficial for the overall performance.

The pre-training has, however, a relatively small impact on the UNREAL method. This might be caused by the ineffectiveness of its auxiliary tasks. As the environment was not continuous, and the rotation actions turned the robot by 90 °, the observations at these rotations were non-overlapping. Therefore, the task of predicting the effect of each action became much more difficult, and instead of helping the algorithm converge faster, it was adding noise to the gradient.

Our additional auxiliary tasks helped the algorithm regardless of whether pre-training was used. We can notice this by comparing our algorithm with UNREAL and 'np ours' with 'np UNREAL' for the no pre-training case. In our method, the auxiliary tasks might help the agent learn a compact representation of each state more quickly and make the algorithm converge faster. We believe that the proposed visual navigation auxiliary tasks have a similar effect as using autoencoders in model-based DRL [37]. In contrast to model-based methods, the agent is not trained to reconstruct the observations precisely. Instead, the image reconstruction loss gradient guides the training process and helps the algorithm converge in the presence of a noisy policy gradient, especially at the beginning of the training.

E. Real-world experiment

The results of the real-world experiments indicate that our algorithm is able to navigate in the real environment well. The discrepancy in the performance between the simulation and the real-world environment can be caused by a generalization error in transferring the learned policy as well as by the errors in odometry measurements, which were used to estimate the robot position for the evaluation of the experiment.

VI. Conclusion & future work

In this paper, we proposed a deep reinforcement learning method for visual navigation in real-world settings. Our method is based on the Parallel Advantage Actor-Critic algorithm boosted with auxiliary tasks and curriculum learning. It was pre-trained in a simulator and fine-tuned on a dataset of images from the real-world environment.

We reported a set of experiments with a TurtleBot in an office, where the agent was trained to find a goal given by an image. In simulated scenarios, our agent outperformed strong baseline methods. We showed the importance of using auxiliary tasks and pre-training. The trained agent can be easily transferred to the real world and achieves an impressive performance. In 86.7 % of cases, the trained agent was able to successfully navigate to its target and stop in the 0.3-meter neighbourhood of the target position with a heading angle difference of at most 30 degrees. The average distance to the goal in the case of successful navigation was 0.175 meters.

The results show that DRL presents a promising alternative to conventional visual navigation methods. Our architecture provides a powerful way to combine a large number of simulated samples with a comparatively small number of real-world ones. We believe that it brings us closer to real-world applications of end-to-end visual navigation, so avoiding the need for expensive sensors.

In our future work, we will evaluate this approach in larger environments with continuous motion of the robot rather than on the grid. We will perform the experiments using a motion capture system for accurate ground-truth localization. To extend the memory of the actor, one can pursue the idea of implicit external memory in deep reinforcement learning [38] and transformers [39]. By using better domain randomization, a general agent can be trained that will not need the robot pose data accompanying the images in the fine-tuning phase. We also plan to work with a continuous action space and with a model of the robot dynamics.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
[2] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, "Agent57: Outperforming the Atari human benchmark," *arXiv preprint arXiv:2003.13350*, 2020.
[3] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.
[4] Y. Wu, Y. Wu, G. Gkioxari, and Y. Tian, "Building generalizable agents with a realistic and rich 3D environment," *arXiv preprint arXiv:1801.02209*, 2018.
[5] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3357–3364.
[6] J. Kulhánek, E. Derner, T. de Bruin, and R. Babuška, "Vision-based navigation using deep reinforcement learning," in *2019 European Conference on Mobile Robots (ECMR)*. IEEE, 2019, pp. 1–8.
[7] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016.
[8] Y. Wu, Y. Wu, A. Tamar, S. Russell, G. Gkioxari, and Y. Tian, "Bayesian relational memory for semantic visual navigation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 2769–2779.
[9] W. Yang, X. Wang, A. Farhadi, A. Gupta, and R. Mottaghi, "Visual semantic navigation using scene priors," *arXiv preprint arXiv:1810.06543*, 2018.
[10] P. Shah, M. Fiser, A. Faust, J. C. Kew, and D. Hakkani-Tur, "FollowNet: Robot navigation by following natural language directions with deep reinforcement learning," *arXiv preprint arXiv:1805.06150*, 2018.
[11] X. Ye, Z. Lin, H. Li, S. Zheng, and Y. Yang, "Active object perceiver: Recognition-guided policy learning for object searching on mobile robots," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 6857–6863.
[12] A. Devo, G. Mezzetti, G. Costante, M. L. Fravolini, and P. Valigi, "Towards generalization in target-driven visual navigation by using deep reinforcement learning," *IEEE Transactions on Robotics*, 2020.
[13] K. Chen, J. P. de Vicente, G. Sepulveda, F. Xia, A. Soto, M. Vázquez, and S. Savarese, "A behavioral approach to visual navigation with graph localization networks," *arXiv preprint arXiv:1903.00445*, 2019.
[14] P. Mirowski, M. Grimes, M. Malinowski, K. M. Hermann, K. Anderson, D. Teplyashin, K. Simonyan, A. Zisserman, R. Hadsell, *et al.*, "Learning to navigate in cities without a map," in *Advances in Neural Information Processing Systems*, 2018, pp. 2419–2430.
[15] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 23–30.
[16] Y.-T. Hu, J.-B. Huang, and A. Schwing, "MaskRNN: Instance level video object segmentation," in *Advances in neural information processing systems*, 2017, pp. 325–334.
[17] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 781–10 790.
[18] W. W. Connors, M. Rivera, and S. Orzel, "Quake 3 arena manual," 1999.
[19] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, *et al.*, "DeepMind Lab," *arXiv preprint arXiv:1612.03801*, 2016.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[21] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[22] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2616–2625.

[23] S. Bhatti, A. Desmaison, O. Miksik, N. Nardelli, N. Siddharth, and P. H. Torr, "Playing Doom with SLAM-augmented deep reinforcement learning," *arXiv preprint arXiv:1612.00380*, 2016.

[24] P. Regier, L. Gesing, and M. Bennewitz, "Deep reinforcement learning for navigation in cluttered environments," 2020.

[25] L. Xie, S. Wang, A. Markham, and N. Trigoni, "Towards monocular vision based obstacle avoidance through deep reinforcement learning," *arXiv preprint arXiv:1706.09829*, 2017.

[26] A. Singla, S. Padakandla, and S. Bhatnagar, "Memory-based deep reinforcement learning for obstacle avoidance in UAV with limited environment knowledge," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–12, 2019.

[27] N. Botteghi, B. Sirmacek, K. A. Mustafa, M. Poel, and S. Stramigioli, "On reward shaping for mobile robot navigation: A reinforcement learning and SLAM based approach," *arXiv preprint arXiv:2002.04109*, 2020.

[28] C. Sampedro, H. Bavle, A. Rodriguez-Ramos, P. de la Puente, and P. Campoy, "Laser-based reactive navigation for multirotor aerial robots using deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1024–1031.

[29] A. V. Clemente, H. N. Castejón, and A. Chandra, "Efficient parallel methods for deep reinforcement learning," *arXiv preprint arXiv:1705.04862*, 2017.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[31] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[32] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: A platform for embodied AI research," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[33] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi, "AI2-THOR: An interactive 3D environment for visual AI," 2019.

[34] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1995–2003.

[35] S. Lowry, N. Sünderhauf, P. Newman, J. J. Leonard, D. Cox, P. Corke, and M. J. Milford, "Visual place recognition: A survey," *IEEE Transactions on Robotics*, vol. 32, no. 1, pp. 1–19, 2016.

[36] Z. Chen, A. Jacobson, N. Sünderhauf, B. Upcroft, L. Liu, C. Shen, I. Reid, and M. Milford, "Deep learning features at scale for visual place recognition," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3223–3230.

[37] N. Wahlström, T. B. Schön, and M. P. Deisenroth, "From pixels to torques: Policy learning with deep dynamical models," *arXiv preprint arXiv:1502.02251*, 2015.

[38] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, 2016.

[39] E. Parisotto, H. F. Song, J. W. Rae, R. Pascanu, C. Gulcehre, S. M. Jayakumar, M. Jaderberg, R. L. Kaufman, A. Clark, S. Noury, *et al.*, "Stabilizing transformers for reinforcement learning," *arXiv preprint arXiv:1910.06764*, 2019.