DEEP REINFORCEMENT LEARNING FOR NAVIGATION IN CLUTTERED ENVIRONMENTS

Peter Regier Lukas Gesing Maren Bennewitz

Humanoid Robots Lab, University of Bonn, Bonn, Germany
{pregier,maren}@cs.uni-bonn.de

ABSTRACT

Collision-free motion is essential for mobile robots. Most approaches to collision-free and efficient navigation with wheeled robots require parameter tuning by experts to obtain good navigation behavior. In this paper, we aim at learning an optimal navigation policy by deep reinforcement learning to overcome this manual parameter tuning. Our approach uses proximal policy optimization to train the policy and achieve collision-free and goal-directed behavior. The output of the learned network are the robot's translational and angular velocities for the next time step. Our method combines path planning on a 2D grid with reinforcement learning and does not need any supervision. Our network is first trained in a simple environment and then transferred to scenarios of increasing complexity. We implemented our approach in C++ and Python for the Robot Operating System (ROS) and thoroughly tested it in several simulated as well as real-world experiments. The experiments illustrate that our trained policy can be applied to solve complex navigation tasks. Furthermore, we compare the performance of our learned controller to the popular dynamic window approach (DWA) of ROS. As the experimental results show, a robot controlled by our learned policy reaches the goal significantly faster compared to using the DWA by closely bypassing obstacles and thus saving time.

1. INTRODUCTION

A prerequisite for nearly all mobile robot applications is collision-free navigation. Typical solutions apply a two-stage approach and use 2D path planning on a cost grid in combination with a low-level motion controller for path tracking and collision avoidance. The low-level controller hereby determines the motion commands for the current time step taking into account the global path and the current robot state as well as the local environment. Typical navigation systems require manual parameter tuning to achieve a good navigation behavior. This tuning requires a significant amount of time and profound knowledge about the navigation software, the robot hardware, as well as the environment conditions, and is a difficult task due to the trade off between time efficiency and safety.

In this paper, we present a self-learned navigation controller realizing collision avoidance and goal-directed behavior. Our approach combines grid-based planning with reinforcement learning (RL) and applies proximal policy optimization (PPO) [1] for the learning task. Our framework hereby uses a global planner to obtain a 2D path from the current robot pose to the global goal. The input of the network consists of the robot's translational and angular velocities, local goals determined from the global path, and a patch of the occupancy grid map containing the obstacles in the robot's vicinity. The outputs are the robot's velocity commands for the next time step. Fig. 1 shows an example situation and a visualization of our approach.

To the best of our knowledge, we present the first solution that integrates global path planning with deep RL to reach a global goal in the environment. Our framework thereby learns the appropriate distance to obstacles and performs regular recomputation of the global path. As a result, no parameter tuning of the navigation controller and inflation of the objects in the map is needed to find the best trade-off between completion time and safety distance to obstacles. When the global

Dhinaharan Nagamalai et al. (Eds): CSEIT, WiMoNe, NCS, CIoT, CMLA, DMSE, NLPD - 2020
pp. 193-204, 2020. CS & IT - CSCP 2020 DOI: 10.5121/csit.2020.101117

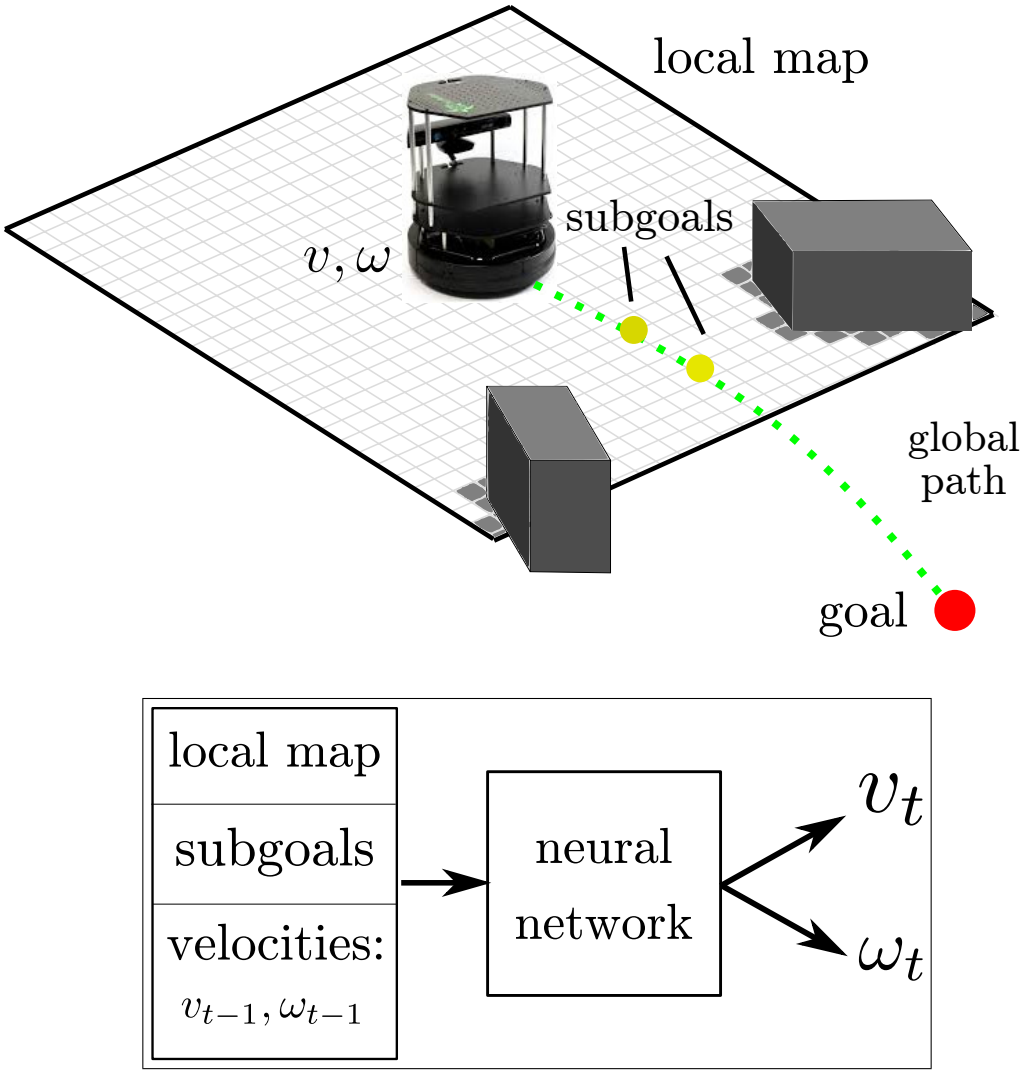

Figure 1: Visualization of our approach, which relies on a global path that is recalculated on a 2D occupancy grid every time step. We use subgoals on the path as part of our observation space and input to a neural network. Additionally, we use a local grid map centered around the robot and the current translational and angular velocities v_{t-1} and ω_{t-1} as network input. We train the network to learn a navigation policy that outputs the velocity commands v_t and ω_t for the next time step.

path leads through narrow passages, where collisions are likely to occur, our system learns to drive around those narrow regions.

We integrated our learned navigation policy as a collision avoidance module that can be used with the Robot Operating System (ROS) navigation stack [2]. We thoroughly evaluated our approach in simulation and in a real-world experiment and compared the performance with the dynamic window approach (DWA) [3], which is used in ROS and which is still one of the most popular navigation schemes. As the experimental results show, the robot steered by our learned policy reaches the goal significantly faster than using the DWA. Furthermore, we show that the policy, which is initially trained only on a simple environment, can be transferred to environments of increasing complexity.

2. RELATED WORK

In the last few years, several learning approaches for mobile robot navigation have been presented. Sergeant *et al.* [4] proposed to use a state representation based on laser range data and learned translational and angular velocity commands for local obstacle avoidance. The authors trained an autoencoder neural network with human-controlled action commands. Pfeiffer *et al.* [5] presented an end-to-end navigation system learned for simple maps using 2D laser data as input, the velocity commands as output, and a 2D path as teacher. The authors later extended the work by applying subsequent RL training to the learned model [6], thus reducing the training time of RL and avoiding overfitting of the imitation model. Liu *et al.* [7] used a local occupancy map as state representation to learn a navigation policy using a variant of the value iteration networks. Tai *et al.* [8] proposed generative adversarial imitation learning to achieve socially complaint navigation. The authors use depth data to train the network and the social force model to generate a large set of training data. Pokle *et al.* [9] designed a local controller to determine the robot's velocity commands and predicting a local motion plan, while considering the trajectories of surrounding humans. These supervised learning methods all depend on the teacher, e.g., controls provided by humans, a global path planner, or a well-tuned optimization, while the goal of our work is to enable the robot to learn by itself while navigating in the environment.

Gupta *et al.* [10] investigated a mapping and planning navigation network based on visual data that encodes the robot's observations into a birds-eye view of the environment, which makes the method limited to known scenarios. Also the approach presented by Hsu *et al.* [11] was developed for known environments. A CNN processes image data and generates discrete actions to move the robot towards a global goal pose. In contrast to that, we use a binary occupancy grid map as representation, which makes the learned policy applicable to environments not seen in the training data.

Chen *et al.* [12] deployed also PPO for deep RL as we do. The authors rely on height-map observations as state representation for a wheel-legged robot. Due to a high-dimensional robot state, the authors discretize the action space and use a set of navigation behaviors to deal with obstacles of certain, given shapes.

Tai *et al.* [13] presented a method that utilizes the robot's velocities and target positions as state representation for an actor-critic RL approach. The authors developed a local controller relying on sparse laser-range measurements and trained a mapless motion planner. Fan *et al.* [14] proposed to use a set of subsequent laser scans and apply PPO to learn movement commands for navigation through crowds. Those approaches do not consider global path planning as they are designed for local navigation.

Chiang *et al.* [15] applied AutoRL to learn two different navigation behaviors, i.e., path following and driving to a global goal location. The authors do not combine learning with global path planning but use the global goal coordinate as input to the network. In our experiments, the robot got stuck in local minima while using only the global goal as input. Therefore, we use a subgoal on the regularly recomputed global path as additional input.

3. PROBLEM DESCRIPTION

We consider a robot moving according to the unicycle model that has to reach a goal location by executing translational and angular velocities. A path planner computes the 2D path to the goal on a global grid map at every time step using the estimated robot pose from a localization system. The RL learning task is to determine the velocity commands for each time step to navigate collision-free and as fast as possible to the goal.

We model the problem as a partially observable Markov decision process (POMDP) defined as

the tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$. Here, $s \in \mathcal{S}$ corresponds to the state of the environment including the robot. The state of the environment changes based on the robot's actions $a \in \mathcal{A}$, which are in our case the translational and angular velocity commands (v, ω), and according to the transition probability $\mathcal{T}(s'|s, a)$. The agent cannot determine the state s but has to rely on observations $\mathcal{O}(o|s', a)$. After every state transition the robot receives a reward $\mathcal{R}(s, a)$.

The actor critic approaches approximate the value function (critic) to be able to update the policy (actor) itself. We use a deep neural network as non-linear function approximator to evaluate the state value function V^{π}, which determines the expected return for state s when following the policy π. The goal of RL is to find a stochastic policy $\pi_\theta(a_t|o_t)$ that maximizes the expected reward

$$\max \mathbb{E}\left(\sum_{k=0}^{T} \gamma^k \mathcal{R}(s_k, a_k)\right),$$

where θ is the set of parameters that specify the function approximator, T is the final time step, and γ is the discount factor.

The critic network is updated based on the advantage value

$$A_t = A(o_t, a_t) = Q^{\pi_\theta}(o_t, a_t) - V^{\pi_\theta}(o_t). \tag{1}$$

where $Q^{\pi_\theta}(o_t, a_t) = r_t + \gamma V^{\pi_\theta}(o'_t)$. Here r_t is the immediate reward at time t and $V^{\pi_\theta}(o'_t)$ is the expected return for the observation o'_t. The actor network uses the policy gradient (PG) method to update the network weights θ in order to maximize

$$\max \mathbb{E}(\log \pi_\theta(a_t|o_t) A_t). \tag{2}$$

Proximal policy optimization (PPO) [1] substitutes the $\log \pi_\theta$ term for the policy probability ratio $\Psi = {}^{\pi_\theta}/_{\pi_{\theta_{old}}}$, to achieve stability. To avoid large policy updates that can impede and reset the training process, the probability ratio is constrained to the range of $[1 - \epsilon, 1 + \epsilon]$ via the $clip$ function

$$\eta^{CLIP}(\theta) = \mathbb{E}_t\left[\min\left(\Psi \mathrm{A}_t, \mathrm{clip}\left(\Psi, 1 - \epsilon, 1 + \epsilon\right) \mathrm{A}_t\right)\right]. \tag{3}$$

4. Neural Network Approximator for Local Navigation

To learn the navigation strategy that takes into account the global path to the goal and the obstacles in the robot's vicinity, we train a deep neural network approximator that provides the robot's translational and angular velocities. The architecture of this network is described in the following.

4.1. Observation Space

The observation space consists of three components as described in the following. The first component is $o_v = (v_{t-1}, \omega_{t-1})$ with v_{t-1} and ω_{t-1} as the robot's current translational and angular velocities computed at the previous time step. The second component is o_m, which corresponds to the $3\,m \times 3\,m$ patch of the 2D occupancy grid map around the robot (see Fig. 2). As resolution of the map we use $0.05\,\mathrm{m}$, thus the grid patch size has a dimension of 60×60 cells.

Additionally, we use a representation of local 2D subgoals in the observation. The subgoal at the current time step is calculated as the position on the global path that is $1m$ away from the robot and stored in map coordinates. At time step t, we transform the global coordinates of the subgoals stored at time steps $t-1$ and $t-5$ into the robot frame to get their relative positions, which serve as third observation component $o_g = (p^x_{-5}, p^y_{-5}, p^x_{-1}, p^y_{-1})$. The representation of the local goal p^x_{-1}, p^y_{-1} indicates the robot's progress that was made towards the goal since the previous time step and is used for the reward calculation. By adding a second subgoal to the observation space $\mathcal{O}$, we noticed an improvement of the navigation policy and speed up of the training. As already noted by Kulhánek *et al.* [16], using information of previous observations helps the system to infer the real state $s \in \mathcal{S}$ of the environment. To summarize, an observation is defined as $o = (o_g, o_v, o_m)$.

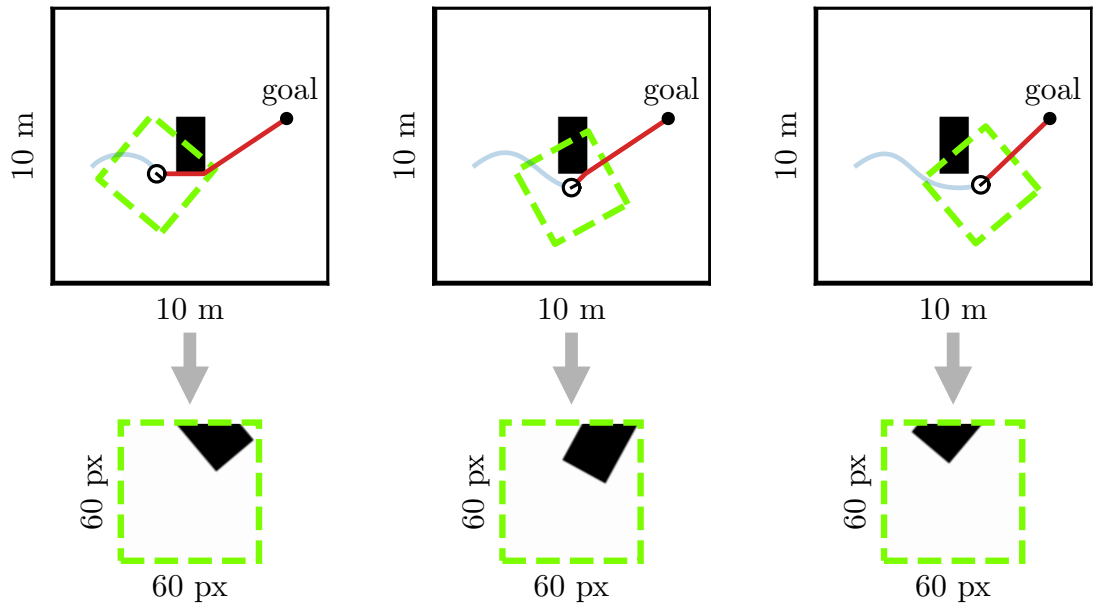

Figure 2: Binary image representation used as input to the network. A $3\,m \times 3\,m$ patch (dashed green) around the robot's pose is cropped from the global occupancy grid map. The robot is at the center of the resulting egocentric image and the viewing direction is to the right side. The global path (red) is computed with the A* search in a binary global map. Interpolated values in the cropped image resulting from the rotation are set to occupied as well as regions outside the boundaries of the global map.

4.2. Reward

Our reward function considers task completion, the duration, and the progress towards the goal

$$\mathcal{R}(s,a) = \mathcal{R}_{fin}(s,a) + \mathcal{R}_{fix} + \mathcal{R}_{dist}(s,a). \tag{4}$$

A navigation task ends if the robot arrives at the goal, a collision occurs, or a maximum number of time steps is reached. Accordingly, the reward $\mathcal{R}_{fin}(s,a)$ is defined as follows:

$$\mathcal{R}_{fin}(s,a) = \begin{cases} b & \text{if the goal was reached} \\ -c & \text{if a collision occurred} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$\mathcal{R}_{fin}(s,a)$ is a large positive value if the distance to the final goal is less than $0.3\,m$, a large negative value if the distance between the robot and the nearest obstacle is less than $0.3\,m$, meaning a collision is occurred, and zero otherwise.

$\mathcal{R}_{fix}$ is a fixed negative reward, that penalizes each action to force the robot to finish an episode as fast as possible.

To speed up the training, we use a third reward component

$$\mathcal{R}_{dist}(s,a) = \alpha \cdot \mathcal{D}_{(s,a)}, \tag{6}$$

where $\mathcal{D}(s,a)$ is the function computing the distance between the robot and subgoal (p^x_{-1}, p^y_{-1}) and α is a scaling factor.

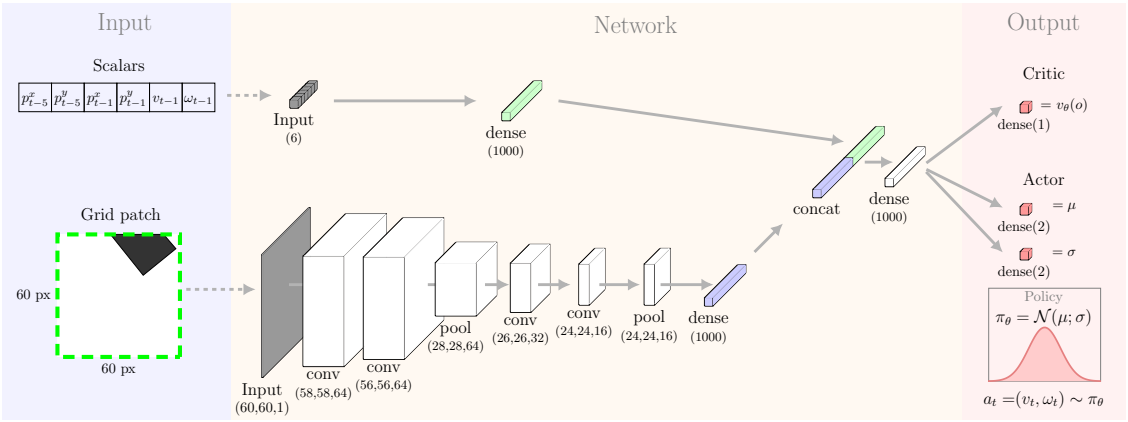

Figure 3: Network structure of the actor-critic scheme. The input consists of scalar values and the grid patch. The scalar values are fed into a single, fully connected dense layer. The binary image of the grid patch (see also Fig. 2) is handled by multiple CNN layers to distinguish obstacle configurations. Then, both branches are concatenated and assembled in a further dense layer. Finally, the critic value $v_\theta(o)$ corresponding to the value function estimator is computed by a last layer. The policy distribution π is calculated by the mean and standard deviation of two normal distributions from which v_t and ω_t are sampled.

4.3. Neural Network Structure

Our observation space as described in Sec. 4.1. is divided based on the representation of the data. Typically, the obstacle grid around the robot is represented as a binary image, while the rest of the observation space provides information about the different components of the robot state. Thus, we propose a network architecture that consist of two branches that split the observation space into scalar values and the binary grid patch (left part of Fig. 3). The scalar branch of the network is a single, fully connected neural network layer (green layer in the upper branch in Fig. 3) and encodes the subgoals and robot velocities into a high dimensional feature space to process them in the following layers.

The grid patch is processed by separate CNN layers (lower branch in Fig. 3), that are well suited for processing 2D data structure, e.g., images. The layers can identify 2D relationships between pixel values and encode obstacles in the robot's vicinity. Max-pooling layers after the first two CNNs reduce the shape and compress the information. This layered design is inspired by the network composition of the well-known VGGnetworks for image recognition [17]. The 3D output of the last max-pooling layer is flattened and reduced to a one-dimensional output with another dense layer (shown in blue). Then, we concatenate the outputs of both branches (blue and green) and process them together in an extra fully connected layer. Finally, we normalize the output, which is a standard technique [18].

The actor and critic estimators share the same connected layers. We found out that the parameter sharing between the actor and critic improves the learning speed because there are fewer parameters to learn. For the value function estimator $v_\theta(o_t)$, the shared network output is inserted into a last dense layer to get a single real number which represents the critic value. The final output of the actor network is the policy $\pi(a|o)$ modeled by the two Gaussian distributions $\mathcal{N}(\mu_{trans}; \sigma_{trans})$ and $\mathcal{N}(\mu_{ang}; \sigma_{ang})$. The two mean values are shrunk with an $tanh$ activation function. This scaling forces the values to stay between the desired velocity limits ($[0 : 0.7]$ m/s and $[-0.7 : 0.7]$ rad/s). The σ values are the standard deviations of the normal distributions. We apply a sigmoid activation function scaled with 0.5 to guarantee that the bandwidths of the normal distributions do not massively grow.

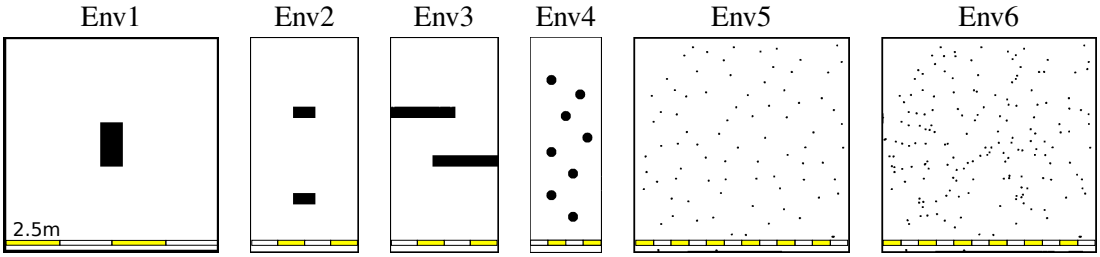

Figure 4: Environments used for training and evaluation. The policy was initially trained only in Env1 and evaluated in all other environments. Afterwards, we used further episodes from Env6 to improve the navigation behavior in highly cluttered scenes as in Env5 and Env6. The evaluation results are depicted in Tab. 1 and Fig. 6.

5. EXPERIMENTS

The implementation of our framework is based on several components. As communication backbone, we use ROS and for the RL approach, we created a simulation environment with *Gazebo* [19]. We implemented the RL in *Python* with the *Tensorflow* library [20]. As mobile platform, we use the Robotino robot by Festo [21].

5.1. Training

To train the neural network and learn a policy to follow a global path and reach a goal without collisions, we used a simple environment (see Env1 in Fig. 4). During the training, we sampled the start and goal positions randomly across the free space, where we chose start-goal configurations with a short Euclidean distance at the beginning and later increased the distance for more challenging scenarios. This helps the robot initially to reach preferable states and learn basic navigation in free space, while longer start-goal configurations force the robot to deal with obstacles, as suggested in [22].

We used four simultaneously operating robots to ensure our collected data is independent and identically distributed. Each robot was given different start and goal configurations. Every episode was limited to 1000 time steps, the batch size was 32 and the entire training involved 10^6 episodes. In Eq. (5), we set the final reward b to 10, c to 50, r_{fix} in Eq. (4) to -0.1, and α in Eq. (6) had value of 10, as experimentally determined. The controller run with a frequency of 10 Hz during training and testing. The overall training time was about 24 hours using a *Nvidia GeForce GTX 1080*.

5.2. Evaluation

After training, we performed experiments in different environments to evaluate the policy learned in Env1 in terms of number of successful runs, which means that the robot reached the goal without collisions, and completion time, both in comparison to the standard ROS [2] navigation stack. The latter uses the DWA [3] to calculate the robot's velocity commands. We configured the DWA with similar restrictions to guarantee similar conditions in terms of acceleration and velocity limits and application of the unicycle robot control. The translational velocity was limited between 0 and 0.7m/s and the angular velocity between -0.7 rad/s and 0.7 rad/s. The acceleration limits for translational and angular steering were set to 1 m/s^2 and 1 rad/s^2 for both approaches.

Note that the DWA approach needs an inflation radius around obstacles in the 2D grid map. This corresponds to a general safety distance to prevent collisions that could result, e.g., from the discretization of the environment. The inflation parameters usually need to be tuned to achieve a good trade-off between safety and time performance. One advantage of our approach is that it

Env1	Env2	Env3	Env4	Env5	Env6	Env5*	Env6*
1.0	1.0	0.99	0.99	0.75	0.22	1.0	0.84

Table 1: Success rate of the trained policy. The evaluation consists of 400 runs for each of the environments shown in Fig. 4. A successful run means that the robot reaches the goal within a certain time limit without any collision. To improve performance in Env5 and Env6, we continued to train the policy on Env6. As shown in the last two columns (Env5* and Env6*), the results of the re-trained policy were seriously better.

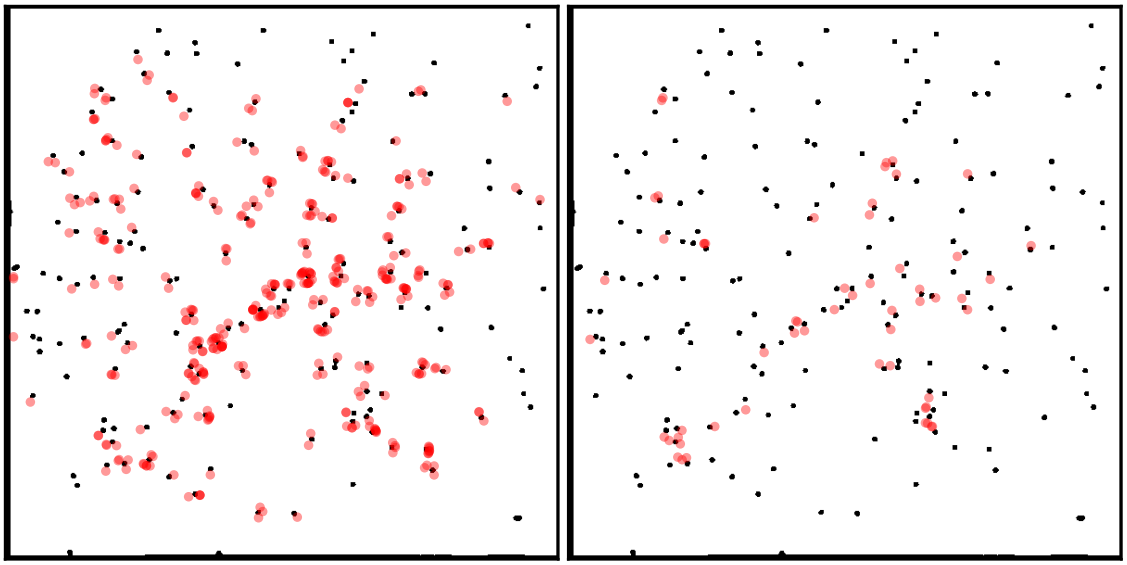

Figure 5: Performance improvement in Env6. (a) Collisions in Env6 with the policy trained on Env1. (b) The policy resulting from the additional training in Env6 shows much fewer collisions.

works on a binary map of the environment without any inflation. Our approach directly learns the appropriate distance to obstacles depending on their local configuration.

Fig. 4 depicts the environments we used in the evaluation. Each map introduces a further level of difficulty. Env2 is similar to the training environment Env1 but the length of the room is doubled and an additional obstacle occurs in the center. The large walls of Env3 can lead the robot into local minima if no global path is used. This map is well suited to test the performance of the learned policy in terms of a reduced completion time while avoiding collisions since fast movements on circular arcs around the obstacle corners are needed to achieve a good navigation behavior. Env4 introduces round obstacle shapes not experienced before. Env5 and Env6 consists of several regions with a high obstacle density. In those maps, it is not always possible to follow the global path computed on a map without obstacle inflation since the path might lead through regions with very close obstacles. Thus, the robot has to learn to bypass the corresponding region by moving away from the global path.

5.3. Success Rate

For each environment in Fig. 4, we performed 400 runs with the DWA and with our trained policy. The robot's start and goal configurations were sampled randomly for each run but were the same for the two approaches. The DWA controller was able to reach all goals in all environments without any collisions. The success rates of our trained policy are listed in Tab. 1. Our approach performs equally well in Env1 to Env4. In Env5 and Env6 the performance decreases due to

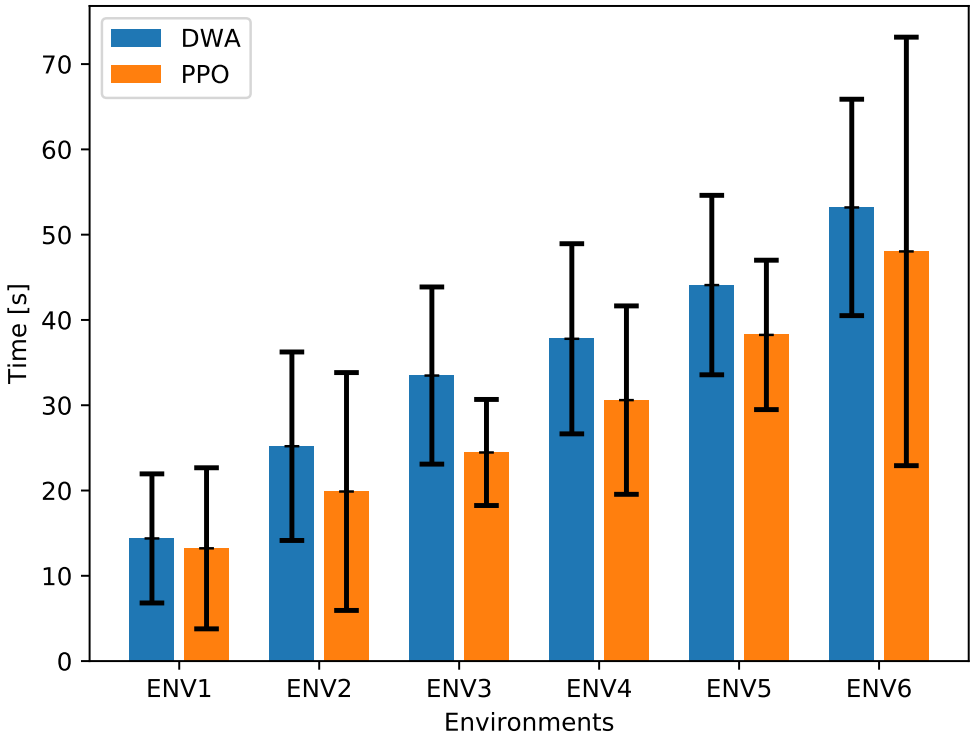

Figure 6: Average completion time for the DWA and our learned policy. The box height shows the average completion time and the whiskers illustrate the standard deviation. As can be seen, our trained policy outperforms the DWA in each environment. The difference is statistically significant in Env3, Env4, and Env5 according to a *paired* t-test at the 0.05 level.

an insufficient generalization resulting from Env1, that leads to increased collision rates. We discovered that situations in which the robot has to depart from the global path did not occur in Env1 and, thus, the robot could not learn a suitable strategy to handle those situations.

To overcome this limitation, we continued the training process with the so far learned policy parameters θ on Env6. After only 8000 further trained episodes (which corresponds to not even 1% of the initial size of the training set), the performance improved significantly and the results are shown in the last two columns of Tab. 1. In Env5 we could achieve a success rate of 100% with the newly learned policy and in Env6 the robot now reached the goal in 84% of all runs (the results are denoted as Env5* and Env6* in Tab. 1).

The left image of Fig. 5 visualizes for Env6 the positions where the robot collided with obstacles when following the policy learned on Env1. The right image of Fig. 5 shows the collisions after further training on Env6. As can be seen, fewer collisions appear in regions with high obstacle density. The reason is that the robot learned when it is beneficial not to follow the global path into narrow space but rather drive around depending on the obstacle configuration.

5.4. Completion Time

Next, we evaluated the completion time of the navigation tasks when using the standard DWA approach and our learned policy. Fig. 6 shows the average completion time for the runs from Sec. 5.3. that were successfully completed by both approaches. Our approach is 16% faster on

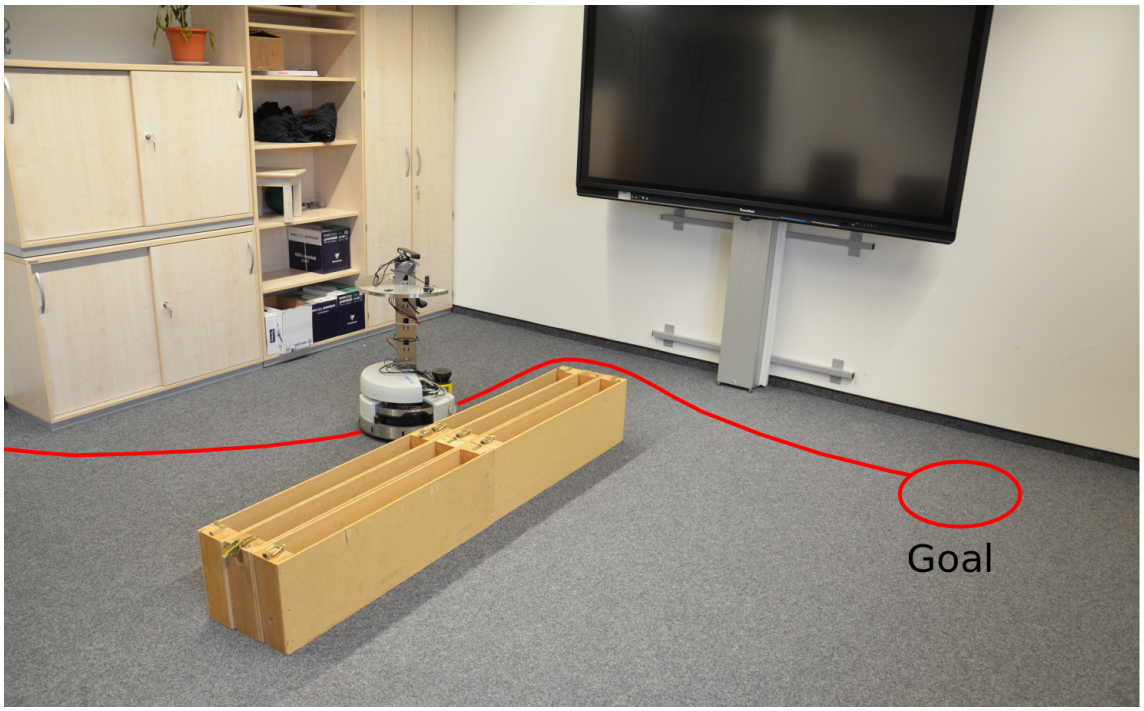

Figure 7: Real-world experiment. The robot entered the office from the left. Based on our learned navigation policy, the robot chooses the best translational and angular velocities to reach the global goal quickly while avoiding the obstacle in the center.

average over all evaluated runs. The difference is statistically significant in Env3, Env4, and Env5 according to a *paired* t-test at the 0.05 level. One reason for the faster performance is that the robot learns the best distance to obstacles, which reduces the trajectory length and leads to time savings, especially in Env3 where our policy performs 26% faster than the DWA.

5.5. Real-World Experiment

Finally, we applied our learned policy on a real robot and compared the performance to the DWA. In the experiment, the robot had to enter an office from the corridor and navigate around an obstacle to reach the global goal (see Fig. 7) by following subgoals on the path. An occupancy grid of the environment was mapped before and we applied Monte Carlo localization [23] to obtain the robot pose.

For the evaluation, we performed 10 experiments with similar start and goal configurations for both the standard DWA approach and our trained policy. With both approaches, the robot reached the goal in each run. The DWA approach needed $28.2\,s$ on average to reach the goal location while our approach had a reduced average completion time of $25.5\,s$. The difference was statistically significant according to a *paired* t-test at the 0.05 level.

Our approach saves time by driving closer around obstacles while the standard DWA takes into account a general inflation radius around obstacles.

6. Conclusions

In this paper, we proposed a new approach to learn a navigation policy for wheeled robots in an unsupervised manner. We use proximal policy optimization for reinforcement learning to train a network that provides the robot's translational and angular velocity commands for the next time

step. Our solution combines global path planning with deep RL to navigate collision-free and reach a global goal in the environment.

Our policy was first trained in a simple environment and subsequently evaluated in environments with increasing complexity. The experimental results demonstrate that our network successfully learned collision-free, goal-directed behavior also in cluttered environments. Furthermore, we compared the performance of our trained policy to the popular dynamic window approach (DWA) with respect to completion time of navigation tasks. On average, the robot controlled by our learned policy completed the tasks 16% faster than the DWA of ROS. In our real-world experiment, we experienced similar results, i.e., the robot performs 10% faster than the DWA using our navigation policy. Our learned strategy safes time by keeping a closer distance to obstacles and choosing appropriate velocities. This is a direct result of the optimization of the motion commands based on the local configuration of the obstacles without any parameter tuning for the navigation controller.

In future work, we plan to incorporate dynamic obstacles, e.g., walking humans, into our framework. An additional sensor would detect the moving obstacles and combine them with the static grid map as input to the CNN.

ACKNOWLEDGMENTS

We would like to thank Christopher Gebauer for helpful discussions and his support during the experimental evaluation. This work has partly been supported by the German Research Foundation under Germany's Excellence Strategy, EXC-2070 - 390732324 (PhenoRob).

REFERENCES

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

[2] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source robot operating system," in *Proc. of the ICRA Workshop on Open Source Software*, 2009.

[3] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine (RAM)*, 1997.

[4] J. Sergeant, N. Sünderhauf, M. Milford, and B. Upcroft, "Multimodal deep autoencoders for control of a mobile robot," in *Proc. of the Australasian Conf. on Robotics and Automation (ACRA)*, 2015.

[5] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[6] M. Pfeiffer, S. Shukla, M. Turchetta, C. Cadena, A. Krause, R. Siegwart, and J. Nieto, "Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations," *IEEE Robotics and Automation Letters (RA-L)*, 2018.

[7] Y. Liu, A. Xu, and Z. Chen, "Map-based deep imitation learning for obstacle avoidance," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[8] L. Tai, J. Zhang, M. Liu, and W. Burgard, "Socially compliant navigation through raw depth inputs with generative adversarial imitation learning," *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[9] A. Pokle, R. Martin-Martin, P. Goebel, V. Chow, H. M. Ewald, J. Yang, Z. Wang,

A. Sadeghian, D. Sadigh, S. Savarese, and M. Vazquez, "Deep local trajectory replanning and control for robot navigation," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[10] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[11] S.-H. Hsu, S.-H. Chan, P.-T. Wu, K. Xiao, and L.-C. Fu, "Distributed deep reinforcement learning based indoor visual navigation," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[12] X. Chen, A. Ghadirzadeh, J. Folkesson, M. Björkman, and P. Jensfelt, "Deep reinforcement learning to acquire navigation skills for wheel-legged robots in complex environments," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[13] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017.

[14] T. Fan, X. Cheng, J. Pan, D. Monacha, and R. Yang, "Crowdmove: Autonomous mapless navigation in crowded scenarios," in *Proc. of the IROS Workshop on From freezing to jostling robots: Current challenges and new paradigms for safe robot navigation in dense crowds*, 2018.

[15] H. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with AutoRL," *IEEE Robotics and Automation Letters (RA-L)*, 2019.

[16] J. Kulhánek, E. Derner, T. de Bruin, and R. Babuška, "Vision-based navigation using deep reinforcement learning," in *Proc. of the Europ. Conf. on Mobile Robotics (ECMR)*, 2019.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. of Int. Conf. on Learning Representations (ICLR)*, 2015.

[18] J. Lei Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint*, 2016.

[19] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2004.

[20] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[21] Festo. [Online]. Available: https://www.festo-didactic.com/int-en/learning-systems/education-and-research-robots-robotino/

[22] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2009.

[23] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo localization: Efficient position estimation for mobile robots," *Proc. of the Conference on Advancements of Artificial Intelligence (AAAI)*, 1999.