Learning fine-grained control for mapless navigation

Fred de Villiers
Applied Mathematics
Stellenbosch University
Stellenbosch, South Africa
20326033@sun.ac.za

Willie Brink
Applied Mathematics
Stellenbosch University
Stellenbosch, South Africa
wbrink@sun.ac.za

***Abstract*—We consider the problem of learning a control policy that allows an autonomous mobile robot to navigate safely to target positions in an environment, without access to an obstacle map. The policy can operate in environments of arbitrary size and may be deployed in resource-constrained settings where storing and maintaining an accurate map are infeasible or prohibitively expensive. The learned policy, trained end-to-end using deep reinforcement learning, outputs continuous control commands to the actuators of a simulated two-wheel differential drive robot. A new reward function is proposed to encourage the robotic agent to learn local recovery and exploration behaviours, which greatly improves the ability of the agent to solve challenging navigation tasks in new environments. The performance of the learned policy is compared to an agent equipped with full knowledge of the obstacle map. Even though the learned policy may solve many navigation tasks, we conclude that some tasks still require the use of a global path planner. However, coupled with a high-level path planner to provide intermediate target beacons to the goal, the learned policy may be employed as an effective low-level component with reactive collision avoidance behaviours and local navigation skills in static or dynamic environments.**

***Index Terms*—autonomous mobile robots, mapless navigation, deep reinforcement learning, continuous control**

I. Introduction

One of the fundamental tasks of autonomous mobile robots is to navigate towards a target position in an environment, without colliding into obstacles. Once a path to the goal has been determined, the robotic agent has to engage the appropriate actuators to execute motion in the desired direction. Conventional methods usually solve these tasks by building a map of the environment. We consider the case where the agent does not have access to a map. Foregoing the use of a map will make certain situations harder to solve. One advantage, however, is that the cost of storing and maintaining a map is removed, which means that the agent may operate in arbitrarily large environments. Many real-world situations also make the construction and maintenance of an accurate map infeasible, for example when the agent is only operational for a short time, or when the environment is highly dynamic.

We assume that an oracle provides the agent with only the position of the target relative to the agent's local reference frame. In practice, this may be calculated with a high degree of precision using GPS coordinates in exterior environments, or WiFi localisation in interior environments [1].

Conventional approaches to navigation depend on human expert knowledge of the dynamics of the environment, and of the locomotive system of the robotic agent. An alternative approach is to apply reinforcement learning, where the agent learns to act appropriately in the environment from past experience, without access to a model of the dynamics of the environment or a human expert's domain knowledge. The reinforcement learning paradigm mimics the dopaminergic system in animals by rewarding the agent when it accomplishes a goal and punishing the agent if it reaches an undesired state [2]. The agent perceives the state of the environment and learns to infer the reward it would expect to receive by taking certain actions, thereby learning the desired control policies to reach the navigation goals. An advantage of this learning-based approach is that the agent may learn general skills that could translate to environments and circumstances unforeseen by a human expert. In addition, once an effective general learning architecture is created it may potentially be reused for different sensor and actuator combinations by simply swapping out the sensorimotor inputs and outputs.

The specific problem we consider is the development of a continuous control policy at the actuator level of a two-wheel nonholonomic differential drive robot equipped with a sparse array of laser rangefinders, to reach a target in the environment without access to (or maintenance of) a map. Environments consist of flat surfaces contained in enclosed spaces, similar to a single floor of an office building.

II. Related work and contributions

Many reinforcement learning approaches simplify matters by sampling from a predetermined discrete set of actions. Instead, Tai et al. [3] trained a simulated robot to take actions in a continuous space in order to maplessly navigate to a target, using deep reinforcement learning [4]. Fan et al. [5] showed that an agent trained with a similar approach can learn effective reactive collision avoidance policies by training in highly dynamic crowded environments. However, the environments and sequence of targets chosen to test these learned policies did not pose much of a challenge.

Zhelo et al. [6] augmented a similar planner with a curiosity model [7] and a limited memory in the form of an LSTM layer in the learning architecture. Their agent was tested in new environments not seen during training, with challenging layout

elements. Extracted features were specific to the training map and did not always transfer to new environments successfully. All of the approaches mentioned rely on a platform-specific controller to translate high-level movement commands, in the form of the linear and angular velocities of the chassis, into actuator activations that control the continuous angular velocities imparted on the wheels of the robot.

Our main contribution is the design of a reward function that aids the agent to learn local recovery and exploration behaviours if there is no clear path to the goal. A further improvement over existing work is that the learned policy will set the angular velocities of the wheels directly. Mechanisms that implicitly learn the features of the training environment will not be considered, since resulting policies often do not transfer well to new environments.

III. Formulation of the learning process

The agent-environment interaction is formulated as a Markov decision process (MDP) [2]. The agent's starting position is determined according to an initial state distribution $p(\mathbf{s}_1)$. At each discrete time step t, the agent observes the state of its environment from the state space $\mathcal{S}$, in the form of k rangefinder readings fanned out at equal angles in front of the agent, denoted by $\mathbf{x}_t = [x_1, x_2, \ldots, x_k]^T$. This is augmented with the distance d_t and angle ϕ_t to the target relative to the agent's local reference frame, along with the current angular velocity of each wheel $\mathbf{a}_{t-1} = [\varphi_L, \varphi_R]^T$, to form the state $\mathbf{s}_t = [\mathbf{x}_t, d_t, \phi_t, \mathbf{a}_{t-1}]^T$. The agent then selects an action $\mathbf{a}_t = [\varphi_L, \varphi_R]^T$, consisting of the angular velocities applied to the left and right wheel, from the action space $\mathcal{A}$. It is assumed that the agent may not move backwards, therefore $\varphi_L, \varphi_R \in [0, \varphi_{\max}]$. The angular wheel velocities result in the linear and angular velocity of the agent's chassis, bringing it to a new position according to the transition distribution $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ of the environment. This distribution is stationary and satisfies the Markov property:

$$p(\mathbf{s}_{t+1}|\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \mathbf{a}_2, \ldots, \mathbf{s}_t, \mathbf{a}_t) = p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t), \quad (1)$$

for any trajectory. Finally, the agent receives a scalar reward to encourage or punish the selected action in the current state according to a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

The agent's goal in the MDP is to select at each discrete time step an action that maximises the expected discounted return. The reward received at time step t is denoted by R_t, and the discounted return G_t is defined as the discounted sum of rewards:

$$\begin{aligned} G_t &\triangleq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \ldots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \ldots) \\ &= R_{t+1} + \gamma G_{t+1}. \end{aligned} \quad (2)$$

The discount factor $\gamma \in [0, 1)$ determines whether the agent only values immediate rewards, when $\gamma = 0$, or whether it takes a longer-term view, as γ approaches 1. A bootstrapped estimate of the recurrence obtained in (2) will serve as a target label during training.

The agent selects an action according to a stochastic policy $\pi_\theta : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ that maps states to probabilities of selecting each possible action, producing a trajectory of states, actions, and rewards: $h_{1:T} = \mathbf{s}_1, \mathbf{a}_1, r_1, \mathbf{s}_2, \mathbf{a}_2, r_2, \ldots, \mathbf{s}_T, \mathbf{a}_T, r_T$. The goal is to learn a policy that maximises the agent's reward. The conditional probability density associated with the policy and parameterised by θ is written as

$$\pi_\theta(\mathbf{a}|\mathbf{s}) = p(\mathbf{a}_t = \mathbf{a} \mid \mathbf{s}_t = \mathbf{s}). \quad (3)$$

The action-value function $q^\pi(\mathbf{s}, \mathbf{a})$ is defined as the expected value of the discounted return, by taking an action $\mathbf{a}$, from a state $\mathbf{s}$, under the policy π_θ:

$$\begin{aligned} q^\pi(\mathbf{s}, \mathbf{a}) &\triangleq \mathbb{E}_\pi \left[G_t \;\middle|\; \mathbf{s}_t = \mathbf{s},\, \mathbf{a}_t = \mathbf{a} \right] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; \mathbf{s}_t = \mathbf{s},\, \mathbf{a}_t = \mathbf{a} \right]. \end{aligned} \quad (4)$$

A good approximation of the action-value function will allow the agent to estimate the best action given any input state, by maximising over $\mathbf{a}$. However, the agent moves by engaging its actuators to impart continuous angular velocities onto the wheels, and maximising $q^\pi(\mathbf{s}, \mathbf{a})$ over a continuous set of actions is computationally infeasible. One option is to discretise the action space, but fine control of actuators needs a fine-grained discretisation in multiple dimensions which may lead to an intractably large set of actions.

To train a policy capable of outputting continuous actions, the deep deterministic policy gradient (DDPG) algorithm [4] is employed. It consists of two deep neural networks: an actor and a critic. The critic network $Q(\mathbf{s}, \mathbf{a} \,|\, \theta^Q)$ is used as a nonlinear function approximator to estimate the action-value function q^π (the Q-value of a given state-action pair), whereas the actor network $\mu(\mathbf{s}|\theta^\mu)$ serves as the policy network, mapping an action to a state. A target network is maintained for both the actor and the critic to ensure stable convergence during training [8], denoted by Q' and μ' with initialised parameters $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$, respectively [4].

State transitions are stored in a replay memory $\mathcal{M}$ and mini-batches of N transitions are sampled uniformly from the memory [8]. For each transition i, the improved estimate of the discounted return is given by

$$y_i = r_i + \gamma Q'(\mathbf{s}_{i+1}, \mu'(\mathbf{s}_{i+1}|\theta^{\mu'}) \,|\, \theta^{Q'}), \quad (5)$$

using the target critic (value) and actor (policy) networks.

The online value network is updated by minimising the mean squared error between the improved estimate of the discounted return and the current estimate of the Q-value:

$$L(\theta^Q) = \frac{1}{N} \sum_i \Big(y_i - Q(\mathbf{s}_i, \mathbf{a}_i \,|\, \theta^Q) \Big)^2. \quad (6)$$

The objective to update the policy network is to maximise the expected return

$$J(\theta) = \mathbb{E}\left[Q(\mathbf{s}_t, \mathbf{a})|_{\mathbf{a}=\mu(\mathbf{s}_t)} \right]. \quad (7)$$

The deterministic policy gradient, or the gradient of the policy's performance, is obtained by applying the chain rule to

calculate the derivative of the objective function with respect to the policy parameter:

$$\nabla_{\theta^\mu} J(\theta) \approx \nabla_{\mathbf{a}} Q(\mathbf{s}_t, \mathbf{a}|\theta^Q)|_{\mathbf{a}=\mu(\mathbf{s}_t)} \nabla_{\theta^\mu} \mu(\mathbf{s}_t|\theta^\mu), \quad (8)$$

as proved by Silver et al. [9]. The policy network is updated by calculating the mean of the policy gradients in the mini-batch:

$$\nabla_{\theta^\mu} J(\theta) \approx \frac{1}{N} \sum_i \nabla_{\mathbf{a}} Q(\mathbf{s}_i, \mathbf{a}|\theta^Q)|_{\mathbf{a}=\mu(\mathbf{s}_i)} \nabla_{\theta^\mu} \mu(\mathbf{s}_i|\theta^\mu), \quad (9)$$

to perform stochastic gradient *ascent* (since the policy seeks to maximise the expected return).

In order to balance stable target labels with accuracy, a soft update is applied to the weights of the target networks to slowly track the online networks during training [4]:

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}, \\ \theta^{\mu'} &\leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}, \end{aligned} \quad (10)$$

where $0 < \tau \ll 1$.

IV. Experimental Design

This section details our simulation environment, the proposed reward function, exploration policies, network architectures, as well the training and testing procedures.

A. Simulator

The use of an existing simulation platform was considered, but a simpler task-specific simulator of our own construction offered easier control of the model parameters and sped up training times. Figure 1 shows the agent and target as circles in the simulated training environment, with the agent's sparse array of rangefinder rays hitting obstacles.

To make the learned policy more robust to possible slippery surfaces or variance in the actuators, noise was sampled from a bivariate Gaussian distribution $\mathcal{N}(\mathbf{0}, \sigma_a^2 I)$, and added to $\mathbf{a}_t$. This noisy input was fed to the kinematic model used by the simulator to form the transition model for training and testing.

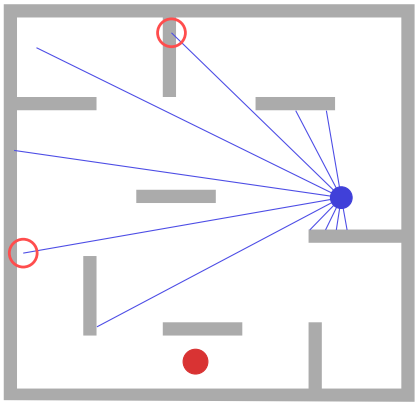


Fig. 1. The simulated two-wheel differential drive robot (blue dot) and target beacon (red dot) in an enclosed training environment. Blue rays represent rangefinder measurements. Input noise may be observed where the rays penetrate beyond the borders of obstacles or do not quite reach obstacles, as highlighted.

Note that the agent has no access to the kinematic model at any stage during the learning or testing process. The agent learns policies by observing and estimating the consequences of engaging its actuators in this unobserved model.

In industrial settings rangefinders are typically accurate to within 1% to 3%. To simulate this, input noise was sampled from a k-dimensional Gaussian distribution $\mathcal{N}(\mathbf{0}, \sigma_s^2 I)$, and added to $\mathbf{x}_t$. All distances were scaled so that each pixel on screen corresponds to one centimetre. The maximum distance measured by the sensors in the simulator was set to $4.0\,\text{m}$. The components of $\mathbf{s}_t$ were normalised to take on values in $[0, 1]$ before being fed into the neural networks.

B. Reward function

Typical reward functions in the literature are variants of the function defined by Tai et al. [3]:

$$r(\mathbf{s}_t, \mathbf{a}_t) = \begin{cases} r_{\text{found}} & \text{if } d_t = 0, \\ r_{\text{crash}} & \text{if } d_{\text{obs}} = 0, \\ \beta_1(d_t - d_{t+1}) & \text{otherwise}, \end{cases} \quad (11)$$

where the agent's distance to the nearest obstacle is denoted by $d_{\text{obs}} = \min_{i\in\{1,2,\ldots,k\}}(x_i)$. The agent receives a positive or negative terminal reward on success or failure, respectively, and an intermediate reward proportional to the change in the distance d_t to the target. The hyperparameter β_1 is usually chosen to scale the magnitude of the intermediate rewards to some fraction of the terminal rewards. This reward function is rich in information, but the agent will be reluctant to move away from the target to negotiate a path around an obstacle if that is the only way to make progress.

We propose a new reward function that does not punish the agent for moving away from the target. Instead the agent is encouraged to learn local recovery and exploration behaviours to find paths around obstacles. The magnitude of the instantaneous linear velocity of the agent is calculated as $v_t = \frac{1}{2}\rho(\varphi_L + \varphi_R)$, where ρ represents the radius of the wheels, and the reward function is defined as:

$$r(\mathbf{s}_t, \mathbf{a}_t) = \begin{cases} r_{\text{crash}} & \text{if } d_{\text{obs}} = 0, \\ r_{\text{unsafe}} & \text{if } d_{\text{obs}} < d_{\text{safe}}, \\ \beta_1 \max\{0, d_t - d_{t+1}\} + \beta_2 v_t & \text{otherwise}. \end{cases} \quad (12)$$

The agent no longer receives a terminal reward for reaching the target. Apart from a terminal reward if the agent collides with an obstacle, there is also a negative reward r_{unsafe} if it moves too close to obstacles. The value of d_{safe} depends on the agent's turning circle. When the agent moves closer to the target it receives a reward proportional to the change in distance to the target, plus a reward proportional to its instantaneous forward speed. Crucially, when the agent moves away from the target it still receives a positive reward proportional to the forward speed. In our experiments, the hyperparameters β_1 and β_2 were chosen so that the maximum positive reward due to moving closer to the target was 0.4 and the maximum positive reward due to speed was 0.2, to ensure that the agent still preferred approaching the target whenever possible.

Agents were trained using both of the reward functions presented above, for comparison. In the sections that follow the purely distance-based reward in (11) will be referred to as the DB reward function, whereas our distance-based reward augmented with a velocity term (12) will be called the DB-V reward function.

C. Exploration during training

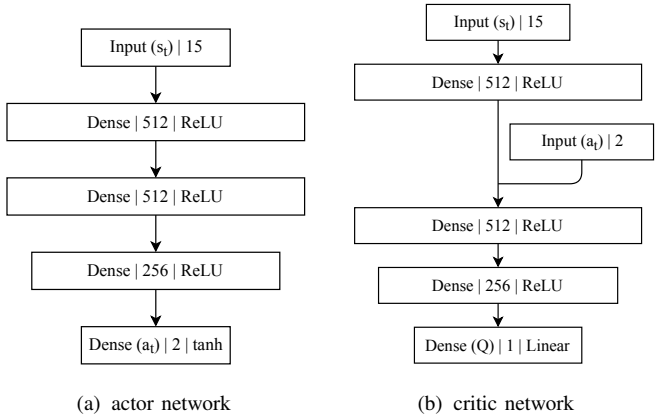
During training, exploration noise is added to the action selected by the actor network $\mu(\mathbf{s}_t|\theta_t^\mu)$. In robotic applications noise is often sampled from an Ornstein-Uhlenbeck process [11], which models the velocity of a massive Brownian particle under the influence of friction. The process is defined by the stochastic differential equation: $dx_t = \theta(\mu - x_t)\,dt + \sigma\,dW_t$, where W_t represents a Wiener process, or Brownian motion. The Ornstein-Uhlenbeck process produces temporally correlated noise that reverts to the long-term mean μ. The values chosen for our experiments were $\mu = 0$, $\theta = 0.15$ and $\sigma = 0.3$.

A separate ϵ-greedy policy was also adapted for the continuous setting, where at each time step the agent takes a random action with probability ϵ. The value of ϵ started at 1.0 and decayed exponentially to reach 10^{-2} by the last frame of training. When the agent selected a random action, the angular velocity of each wheel was sampled uniformly from $[0, \varphi_{\max}]$. Agents were trained using both of the exploration strategies for comparison.

D. Learning architecture and hyperparameter selection

The architectures of the actor and critic networks are presented in Figure 2. Both have three fully-connected hidden layers with rectified linear unit (ReLU) activations to introduce nonlinearity. Output values from the actor were constrained to the range $(-1, 1)$ by using the hyperbolic tangent $(\tanh)$ as activation. The outputs were then linearly transformed to the range $(0, \varphi_{\max})$.

The weight matrices and bias vectors of the hidden layers of both networks were all initialised in the same way, similar to the scheme presented in [4]. If n is the dimension of the input to a layer, the layer's weights and biases were sampled uniformly from $[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$. Initial weights and biases for the output layers of both the actor and the critic were sampled uniformly from $[-3\times10^{-3}, 3\times10^{-3}]$ to ensure that the initial Q-value estimates and velocities were close to zero.



(a) actor network (b) critic network

Fig. 2. The structures of the actor and critic networks, each consisting of three fully-connected hidden layers.

The networks were trained using the Adam optimiser [12] with a learning rate $\alpha = 10^{-4}$ and mini-batches of size 32. The discount factor γ was set to 0.99. The soft update to the target networks was implemented with $\tau = 10^{-3}$. It was not necessary to perform any hyperparameter optimisation or grid searches. Reasonable values based on similar work were used in all cases. Smaller networks with fewer units per layer were used for validation purposes, and then expanded trivially until they produced smooth policies.

E. Training procedure

Four agents were trained for the final experiments: two received the DB reward (11), and two received the DB-V reward (12). One agent from each group followed an ϵ-greedy exploration strategy, while the other sampled exploration noise from an Ornstein-Uhlenbeck (OU) process [11]. The four agents are labelled DB-ϵ, DB-OU, DB-V-ϵ, and DB-V-OU in the discussions that follow.

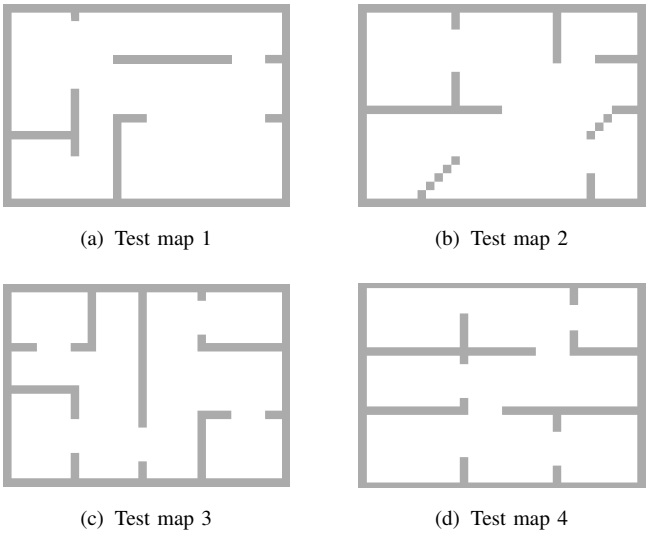
The replay memory $\mathcal{M}$ for storing transitions had a capacity $c = 10^6$. An initial set of $50,000$ random transitions were sampled in the training environment to pre-fill $\mathcal{M}$.

All of the agents were trained for 7.5 million frames in the training environment presented in Figure 1. The environment was chosen to pose a diverse set of challenges during training. Both the agent and the target were spawned at random positions at the start of each episode. If the agent reached the target, a new beacon was spawned randomly in the environment and the agent had to find the new target from its current position. A training episode would end after a maximum of $5,000$ frames, or as soon as the agent collided with an obstacle.

F. Testing procedure

The performance of the fully trained policies were evaluated not only in the training environment, but also in the four test environments presented in Figure 3, to gauge whether the learned policies could transfer to new environments. These test maps, adapted from [6], are arranged more-or-less in order of difficulty and confront the agent with structural elements not encountered during training. Each contains potential starting positions that will require the agent to move away from the target in order to move around an obstacle. Some of the maps have narrower doorways to enclosed areas. Test map 2 has diagonal barriers not present in the training map. Test maps 3 and 4 have long wall sections that may require a long detour to negotiate.

For each map, 300 random starting and target position pairs were generated and all the trained policies were tested on this fixed set of problems. The agents were given a maximum of $2,500$ steps to reach each of the target positions. The number of steps taken was recorded for each problem as well as the success rate of reaching the targets.



(a) Test map 1 (b) Test map 2

(c) Test map 3 (d) Test map 4

Fig. 3. The performance of the trained policies were tested in four test environments offering challenges not encountered in the training environment.

V. Results

The performance of the policies were compared to a global path planner that had full access to the obstacle map. The global path planner discretised the map and performed A* search [13] to find the shortest path to the goal on a grid. A DB-V-ϵ agent was augmented with the global path planner by providing the agent with intermediate targets along the shortest path to the final goal. This agent is labelled DB-V-Global.

Table I summarises the performance of the final policies evaluated in the training and test environments, and the success rate of each policy is plotted in Figure 4. All of the policies fared well in the training environment and could solve almost all of the tasks. The drop-off in performance of the mapless policies in the test environments is expected, as each test map contains potentially adversarial situations that are difficult for a mapless policy to solve. The policies trained with our DB-V reward function are clearly better than the DB policies at reaching the targets, attaining success rate differentials of between 8% and 23% depending on the difficulty of the tasks. This translates into an increase in performance of 10% to 40% over the DB policies.

The performance of the DB-V-OU and DB-V-ϵ policies are very similar in the first three test environments. In the last and most challenging test environment, however, the DB-V-ϵ policy is clearly better. The fact that policies trained with ϵ-greedy exploration perform better than their OU counterparts may indicate that better values need to be found for the parameters controlling the exploration noise. It may also indicate that the usual OU-based exploration strategies employed for policy gradient methods may not always be optimal. In any case, the ϵ-greedy policy may work well with the two-dimensional action space considered here, but may not be as effective for systems with more degrees of freedom.

Next we consider an instructive example in Figure 5, which illustrates the main difference between the learned policies

TABLE I
Evaluation in training and test environments.

Environment	Reward-Exploration	Success rate (%)	Steps (mean $\pm$ stddev)
Train map	DB-OU	86.67	568.50 ± 44.44
	DB-ϵ	92.33	450.64 ± 35.81
	DB-V-OU	99.00	298.07 ± 15.39
	DB-V-ϵ	99.67	285.87 ± 11.13
	DB-V-Global	100.0	288.90 ± 13.22
Test map 1	DB-OU	75.67	831.12 ± 55.96
	DB-ϵ	80.67	709.96 ± 51.63
	DB-V-OU	90.33	530.83 ± 38.68
	DB-V-ϵ	90.67	514.33 ± 38.57
	DB-V-Global	99.33	327.62 ± 15.54
Test map 2	DB-OU	78.33	723.30 ± 53.60
	DB-ϵ	82.67	634.28 ± 49.92
	DB-V-OU	90.67	497.71 ± 39.58
	DB-V-ϵ	93.00	432.02 ± 34.37
	DB-V-Global	99.67	264.36 ± 8.39
Test map 3	DB-OU	55.67	1257.77 ± 64.33
	DB-ϵ	65.00	1096.84 ± 61.46
	DB-V-OU	86.33	731.03 ± 45.15
	DB-V-ϵ	84.00	747.87 ± 46.90
	DB-V-Global	97.67	410.98 ± 22.54
Test map 4	DB-OU	57.33	1227.33 ± 64.15
	DB-ϵ	59.67	1184.91 ± 63.67
	DB-V-OU	74.33	881.53 ± 56.24
	DB-V-ϵ	83.00	746.85 ± 48.62
	DB-V-Global	99.33	351.56 ± 13.65

using the various reward functions. The DB-V-Global agent finds a near-optimal trajectory to the target, whereas both of the mapless agents are tricked into a blind alley as they turn the first corner at the top-right. The DB-ϵ agent is reluctant to move away from the target and remains stuck in the corner. The DB-V-ϵ agent, however, keeps exploring locally and in this case loops back far enough to find its way to the target.

Finally, a task was designed to illustrate the limits of the local recovery behaviour learned by the DB-V-ϵ policy (refer to Figure 6). The DB-V-Global policy once again finds a near-optimal path to the target, despite a slightly coarse trajectory at the end due to the discretisation inherent in the global path planner. The mapless DB-V-ϵ agent tries to explore locally, but

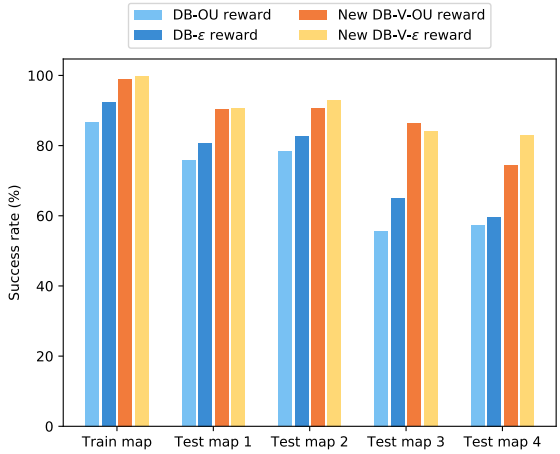
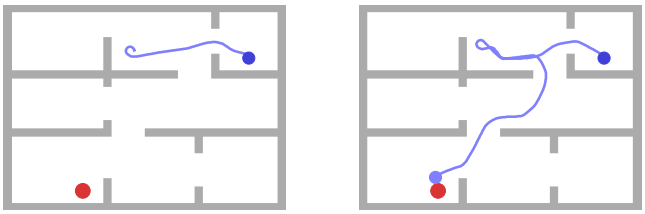
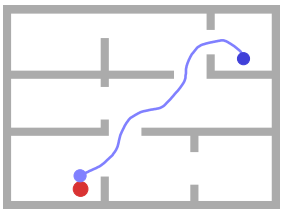


Fig. 4. The success rate of each policy in the training and test environments.

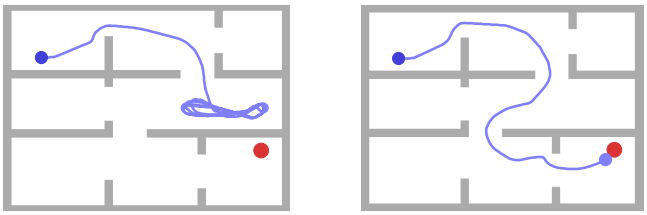


(a) DB-ϵ agent (b) DB-V-ϵ agent



(c) DB-V-Global agent

Fig. 5. An adversarial example in the challenging Test map 4 where the mapless agents are lead into a blind alley. The DB-ϵ agent remains in the corner unwilling to move away from the target, whereas the DB-V-ϵ agent keeps exploring locally to loop back and find a path to the target.



(a) DB-V-ϵ agent (b) DB-V-Global agent

Fig. 6. Another adversarial example in Test map 4. This time the DB-V-ϵ agent keeps exploring locally but cannot find a path to the target, searching back and forth in a figure-of-eight pattern.

without knowledge of the map it turns back toward the target before it can find an opening in the wall section and continues to move in a figure-of-eight until the end of the episode.

VI. Conclusion

Our objective was to train a robotic agent equipped with a sparse array of rangefinders to navigate safely to targets in an environment, without access to an obstacle map or knowledge of the dynamics of the environment. The agent was trained end-to-end using deep reinforcement learning and the learned control policy had to output continuous control commands at the actuator level while expected to execute high-level pathfinding behaviours. A new reward function was proposed that augmented the reward signal with a term based on the velocity of the agent, encouraging the agent to learn local recovery and exploration behaviours and dramatically improving the agent's ability to solve difficult navigation problems in challenging, previously unseen environments.

The mapless control policy is not designed to replace map-based approaches. A few simple adversarial examples were considered that were trivial for a global map-based path planner, but were impossible for the mapless policy to solve. However, the learned policy exhibited useful reactive collision avoidance and local navigation skills and may constitute an effective low-level component for continuous control, coupled with a coarser global path planner. The agent that was developed to serve as a benchmark for the mapless policies is an illustration of exactly such a system. Futhermore, the learned policy may also be deployed effectively in resource-constrained systems where the cost of maintaining a map is prohibitive, or in situations where the construction and maintenance of a map are infeasible.

The learning architecture presented here extends naturally to dynamic environments by simply training the agent in an environment with moving obstacles. Future work may consider augmenting the model with a mapping component that may allow the agent to discern when it is not making progress toward the target. A more advanced learning architecture may be developed that allows the agent to leverage its knowledge of the map to autonomously generate intermediate waypoints to the target, but care must be taken so that the policies learned from the training environment are general enough to transfer to new environments.

References

[1] Y. Sun, M. Liu, and M. Q.-H. Meng, WiFi signal strength-based robot indoor localization, *IEEE International Conference on Information and Automation*, pp. 250–256, 2014.

[2] R. S. Sutton, and A. G. Barto, Reinforcement learning: an introduction. Cambridge, MA: The MIT Press. 2018.

[3] L. Tai, G. Paolo, and M. Liu, Virtual-to-real deep reinforcement learning: continuous control of mobile robots for mapless navigation, *IEEE International Conference on Intelligent Robots and Systems*, pp. 31–36, 2017.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, Continuous control with deep reinforcement learning, *International Conference on Learning Representations*, 2016.

[5] T. Fan, X. Cheng, J. Pan, D. Manocha, and R. Yang, CrowdMove: autonomous mapless navigation in crowded scenarios, *arXiv preprint*, arXiv:1807.07870, 2018.

[6] O. Zhelo, J. Zhang, L. Tai, M. Liu, and W. Burgard, Curiosity-driven exploration for mapless navigation with deep reinforcement learning, *arXiv preprint*, arXiv:1804.00456, 2018.

[7] D. Pathak, P. Agrawal, and A. Efros, Curiosity-driven exploration by self-supervised prediction, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2017.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, s. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, Human-level control through deep reinforcement learning, *Nature*, pp. 529–533, 2015.

[9] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, Deterministic policy gradient algorithms, *International Conference on Machine Learning*, pp. 387–395, 2014.

[10] Robotis, Turtlebot 3 specifications, *Robotis e-manual*, Retrieved from http://emanual.robotis.com/docs/en/platform/turtlebot3/specifications, 2019.

[11] G. E. Uhlenbeck and L. S. Ornstein, On the theory of Brownian motion, *Physical Review*, vol. 36 (5), pp. 823–841, 1930.

[12] D. P. Kingma and J. L. Ba, Adam: a method for stochastic optimization, *International Conference on Learning Representations*, 2015.

[13] S. J. Russel, and P. Norvig, Artificial intelligence: a modern approach. Upper Saddle River, New Jersey: Prentice Hall. 2010.