Entropy-Based Exploration for Mobile Robot Navigation: A Learning-Based Approach

Nicolò Botteghi[1], **Khaled Alaa**[2], **Beril Sirmacek**[3], **Mannes Poel** [4]

Abstract

This work presents an exploration strategy for a Deep Reinforcement Learning path planner for learning continuous velocity commands from raw sensory data for solving navigation tasks in indoor environments. To the best of the authors' knowledge, this is the first approach in which the map's entropy, built online by the robot using a Simultaneous Localization and Mapping algorithm, is utilized during the training phase to shape the reward function. The results show that the entropy term of the reward function motivates the robot to improve its exploration capabilities and makes it able to escape local minima in environments with relatively complex topology. Additionally, the proposed map-less planner has achieved a comparable performance compared to a traditional motion planner that requires a precise map of the environment beforehand. The effectiveness of the proposed approach is verified by the successful generalization of the learned policy to previously unseen environments. A video of our experiments can be found at https://youtu.be/QWRyocjQzzw

Introduction

Autonomous navigation in unknown and complex environments is an important challenge for robotics. This problem is usually tackled using path planning algorithms (i.e potential field, cell decomposition, *A* graph search* (Stentz 1994)) by relying on representations of the environments: the maps. These maps are usually built using Simultaneous Localization and Mapping (SLAM) algorithms (Thrun, Burgard, and Fox 2005). However, in many interesting applications, a complete map of the environment is expensive to obtain or difficult to keep up-to-date.

In recent years, Deep Reinforcement Learning (RL) (Sutton and Barto 1998) has been used to tackle and solve several different robotics tasks such as stabilization, manipulation, locomotion and navigation. In the context of navigation, path planners based on RL don't usually rely on any map or SLAM and, even though very successful, they don't exploit any of the important information stored in the maps. To this category belongs the work presented in (Tai, Paolo, and Liu 2017), (Zhelo et al. 2018), (Pfeiffer et al. 2017), (Duo et al. 2019) and (Zhang, Zhang, and Liu 2018) where map-less RL path planners are developed in order to solve target-reaching navigation tasks in unknown environments with static or simple dynamic obstacles configurations.

On the other hand, few approaches have tried to combine the RL planners with maps and SLAM. In this category, the work in (Zhang et al. 2016) proposes a successor feature Deep Q-Network algorithm for solving navigation tasks when a map of the environment is known a priori. In (Brunner et al. 2017), Asynchronous Advantage Actor-Critic (A3C) is used to navigate the robot out of a random maze of which the map is given. The observations of the agent include 2D-map of the environment, the robot's heading direction, the previously estimated pose and the actions taken. In (Zhang et al. 2017), an external memory acting as an internal representation of the environment for the agent is fed as an input to an RL algorithm (A3C). The agent is thus guided to make informed planning decisions to effectively explore new environments. However, these methods suffer from the need to store the map information not only during training, but also during the deployment and testing phase. Especially in the deployment phase on real robots, the computational power and memory may be limited and computing and storing a full map might be prohibited. Furthermore, all the approaches in this category rely on a discrete action space for the agent (e.g. move forward, backward, left and right).

To address this problem, in (Mustafa et al. 2019) and (Botteghi et al. 2020), we combined an RL path planner with a reward function based on the map built with a SLAM algorithm. The map is only used during training to improve the learning performances and the navigation skill, however, it is not used during testing. The policies learned with this approach are not only able to generalize well to unseen environments and targets, but they can be directly, without any further tuning, transferred to the real robot as well. Furthermore, we rely on a continuous action space to achieve more

[1] Robotics and Mechatronics, Faculty of Electrical Engineering, Mathematics and Computer Science, CTIT Institute, University of Twente, The Netherlands, n.botteghi@utwente.nl

[2] Intelligent Driving Functions, IAV GmbH (Volkswagen Group), Germany, khaled.mustafa@iav.de

[3] Jönköping AI Lab (JAIL), School of Engineering, Jönköping University, Sweden, beril.sirmacek@ju.se

[4] Datamanagement and Biometrics, Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, the Netherlands, m.poel@utwente.nl

advanced maneuvers and smoothness of the trajectories.

In our previous work, (Mustafa et al. 2019) and (Botteghi et al. 2020), we only considered the distance to the obstacles to achieve higher obstacle-awareness. However, the improved collision-awareness alone, granted by those methods, is not enough when the environments grow in complexity (e.g. multi-room indoor environments) and present local minima (e.g. getting stuck in a room with the target on the other side of the wall). In this work, we further strengthen the connection between RL and SLAM by exploiting more of the knowledge stored in the map. In particular, we use the map's entropy to improve the exploration skills of the planner and its ability to escape navigation minima that occur when the environments are more complex and realistic. The proposed method is shown in Figure 1.

The rest of the paper is organized as follows. In Section *Background*, the theory behind Reinforcement Learning and Simultaneous Localization and Mapping is presented. Our proposed approach is described in details in Section *Methodology*. Section *Experimental design* describes the experiments performed. Furthermore, Sections *Results and discussion* and *Conclusion* discuss the results and the conclusions.

Background

Reinforcement Learning

RL (Sutton and Barto 1998) is the Machine Learning (ML) branch in charge of learning sequential decision making processes. The agent, i.e. the decision-maker, by interacting with the environment tries to learn the optimal way of behaving. This interaction process can be modelled as a Markov Decision Process (MDP), $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is a set of actions, $\mathcal{P}(s_{t+1}|s_t, a_t)$ is the state-transition probability distribution and $\mathcal{R}(s_t, a_t)$ is the reward function. The aim of RL is finding the optimal policy $\pi_\theta(s_t)$, mapping states into actions, for maximizing the total cumulative discounted rewards in Equation (1).

$$R = \Sigma_{t=0}^{T} \gamma^t r_{t+1} \tag{1}$$

Mobile robot navigation can be phrased as an RL problem in which the goal is to learn collision-free paths to reach target locations in environments with unknown topology and obstacle configuration.

Deep Deterministic Policy Gradient Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al. 2016) is a model-free, off-policy RL algorithm with an actor-critic structure. The actor, usually represented by a neural network, chooses an action a_t for each given state s_t. On the other hand, the critic, represented as well by a neural network, assesses the performances of the actor by estimating the state-action value function Q as it happens in the Deep Q-Network (DQN) algorithm (Mnih et al. 2013). DDPG can be seen as the extension of DQN for continuous action spaces.

The parameters of the critic network, θ^Q, are adjusted according to Equation (2).

$$L_i(\theta_i^Q) = E_{s\sim\rho_\pi, a\sim\pi}[(Q(s_t, a_t|\theta_i^Q) - y_i)^2] \tag{2}$$

where $y_i = r(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1}|\theta_i^Q)$ is the the target Q-value used for computing the error that is backpropagated to adjust the parameters of the critic network and π is the behavioral policy that the robot uses for exploration. The estimation of the state-action value function Q is used to update the parameters of the actor network, θ^π, that are adjusted in the direction of the gradient of the expected return, $\theta_{i+1}^\pi = \theta_i^\pi + \alpha_{\theta^\pi} \nabla_{\theta^\pi} J(\pi_\theta)$. The gradient of the expected return is shown in Equation (3).

$$\nabla_{\theta^\pi} J(\pi_\theta) = E_{s\sim\rho_\pi}[\nabla_a Q(s_t, \pi(s_t|\theta^\pi)|\theta^Q) \nabla_{\theta^\pi} \pi(s_t|\theta^\pi)] \tag{3}$$

In this way, the improvements in the actor's policy are guided by the estimated state-action value function.

Simultaneous Localization and Mapping

In order to increase the robot's spatial awareness, we benefit from a SLAM algorithm. SLAM algorithms have been well investigated for more than a decade in both computer vision and robotics communities. Depending on the sensors, map building needs, timing requirements, computation platforms, researchers proposed different SLAM algorithms. Nevertheless, all SLAM algorithms are developed for building a 2D or a 3D map of the environment around the robot while at the same time finding the relative pose (location and orientation vectors) of the robot within this environment. While the robot is moving in the environment, the SLAM algorithm is iterated. For each time step, the next time step robot pose is estimated and also robot pose measurement is performed. The robot pose measurement is compared with the estimation done in the previous step and the difference (error) is used for updating the system parameters to perform better estimations (Thrun, Burgard, and Fox 2005).

In our study, we used Rao-Blackwellized particle filter (RBPF) to solve the estimation step of the SLAM cycle. The RBPF method separates the estimation of the robot's pose from the posterior of the environment map (Murphy 1999) as shown in equation (4):

$$p(x_{1:t}, m|z_{1:t}, u_{1:t-1}) = p(m|x_{1:t}, z_{1:t}) p(x_{1:t}|z_{1:t}, u_{t-1}) \tag{4}$$

where $x_{1:t}$ represents the robot's trajectory, $z_{1:t}$ is the set of observations, u_{t-1} is the control input and m defines the built map of the environment.
The advantage of using this factorization is that the estimation of the joint posterior can be divided into two separate steps:

Particle Filter Estimation. The particle filter estimates the robot's pose $p(x_{1:t}|z_{1:t}, u_{t-1})$, represented as a probability distribution due to the presence of uncertainties in the robot's motion and measurements, through a finite set of particles (5).

$$\mathcal{X}_t := \{x_t^{(1)}, x_t^{(2)}, ..., x_t^{(N)}\}, \tag{5}$$

where each particle $x_t^{(n)}$ represents a belief of the true state at time step t. Moreover, the new set of particles $\mathcal{X}_t$ is constructed recursively from the previous set $\mathcal{X}_{t-1}$ by sampling from a proposal distribution, i.e. the probabilistic motion model $p(x_t^{(n)}|x_{t-1}^{(n)}, u_{t-1})$, usually built using odometry information. Then, the probabilistic observation model

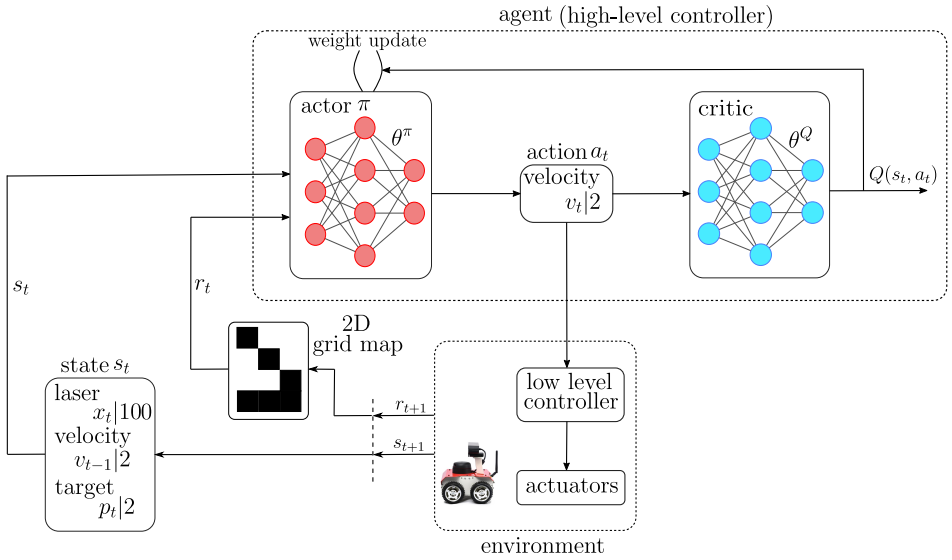


Figure 1: Proposed architecture for continuous control navigation in unknown environment. The motion planner is trained through a deep-RL within an off-policy, actor-critic framework that represents the high-level controller of the robot using only sparse laser data. The robot's low level controller executes the navigation actions determined by the motion planner. The occupancy grid map is built online by the robot and used to shape the reward function.

$p(z_t|x_t^{(n)})$ is incorporated and an importance weighting factor $w_t^{(n)}$ is assigned to each particle. Through selective resampling (Grisetti, Stachniss, and Burgard 2005), the particles with least importance weights, correspondent to unlikely robot's poses, are neglected. Eventually, this iterative procedure converges to the correct estimate.

Mapping with Known Poses. After computing the pose estimate, it is possible to estimate the posterior of the map $p(m|x_{1:t}, z_{1:t})$. This step is usually called *mapping with known poses* (Moravec 1988). The map, in this context, is represented using an occupancy grid in which each cell can be either occupied, unoccupied, or unknown. The resolution of the grid cells should be chosen to be compatible with the smallest feature of the environment. When dealing with occupancy grid maps, the probability of every grid cell, whether it is occupied or not, is assumed to be independent of the others. By making this assumption, the posterior probability of the entire map can be computed as the product of the posterior of every grid cell m_i in the map, as shown in Equation (6).

$$p(m|z_{1:t}, x_{1:t}) = \prod_{i=0}^{M} p(m_i|z_{1:t}, x_{1:t}) \tag{6}$$

where M is the total number of cells in the map.

Methodology

Approach

In this work, we present the development of an RL-based motion planner. The RL-agent uses only raw sensory information for determining the sequence of continuous velocity commands (linear and angular velocities), that the robot has to execute to reach target locations in environments with unknown obstacle configuration. Differently from standard map-less RL motion planners, only during the training phase, we exploit the knowledge of the environment stored in the map, built online using the RBPF SLAM algorithm introduced in Section *Background*, to improve the training speed and navigation skills of the agent. The incorporation of the knowledge stored in the map happens through the shaping of the reward function. Unlike many RL navigation approaches that employ only target driven reward functions based on the distance to the target with penalties for hitting the obstacles, to aid the navigation in complex environments, we proposed a new reward function including the map's entropy to encourage exploratory behaviors and, hopefully, escape local minima.

Reward Function

The agent's goal is to generate velocity commands to control a mobile robot. The robot has to be able to reach the desired targets while avoiding collisions with obstacles and escape local minima. To achieve these three objectives, the proposed reward function is equal to the weighted sum of three different terms:

- Target Driven (TD)
- Obstacle Awareness (OA)
- Map's Entropy (Entropy)

The TD term of the reward function is based on the Euclidean distance robot-target. In particular, the difference of the distance to the target at the current time step and the distance to the target at the previous time step (see Equation (7)). Intuitively, the agent is positively rewarded if it gets

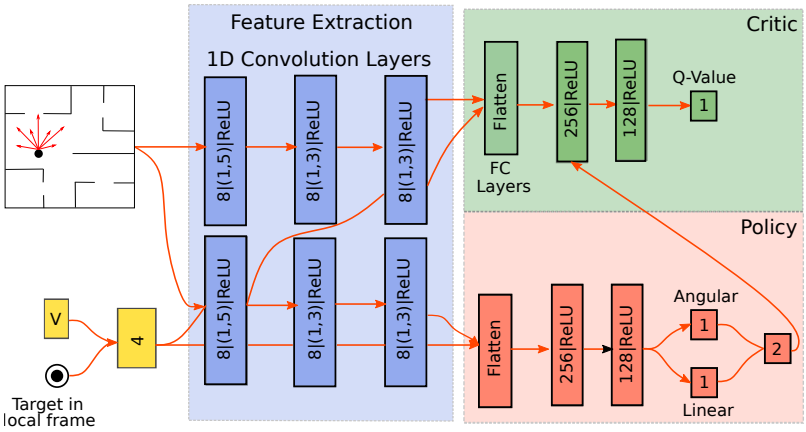


Figure 2: The laser data is processed by the feature extraction part which consists of three convolution layers. The FC part of the network fuses the extracted features and the target information.

closer to the target.

$$r_1(s_t) = \left\|\mathbf{p}_{t-1}^{x,y} - g\right\|_2 - \left\|\mathbf{p}_t^{x,y} - g\right\|_2 \tag{7}$$

where $\|p_t^{x,y} - g\|_2$ is the distance from the current position p_t of the robot at time t with respect to the inertial frame and the target's location g expressed in the robot's coordinate frame.

The OA term of the reward function includes a penalty inversely proportional to the squared distance to the obstacles (see Equation (8)). It is worth to mention that the distance to the obstacle is computed online using the laser range (i.e. LiDAR) data and the position of the obstacles is not assumed to be known a priori. This term of the reward function is activated once the distance to an obstacle, measured by the LiDAR, is smaller or equal than 0.6m. The agent is negatively rewarded if it gets closer to obstacles.

$$r_2(s_t) = \frac{1}{\left(\left\|\mathbf{p}_t^{x,y} - o\right\|_2\right)^2} \tag{8}$$

The third term is the map's entropy term (see Equation (9)). The map's entropy corresponds to the sum of the entropy of all the cell c in the map m. The agent is rewarded if it explores the environment or navigates through open and more uncertain areas.

$$r_3(s_t) = \sum_{c_f \in m} p(c)\log p(c) + \sum_{c_o \in m} (1-p(c))\log(1-p(c)) \tag{9}$$

where c_f corresponds to the unknown and unoccupied cells in the map and c_o to the occupied ones.

In addition, a sparse reward, r_{reached}, is added if the agent reaches the target within a predefined distance threshold and a penalty, r_{crashed}, if the robot either collides with an obstacle or exceeds the maximum number of steps T in a single episode. Based on the above considerations, the overall reward function $r(s_t)$ in shown in Equation (10).

$$R(s_t, m) = \begin{cases} r_{\text{reached}}, & d \leq d_{min}, \\ r_{\text{crashed}}, & s_{ts}, \\ \lambda^g r_1(s_t) - \lambda^o r_2(s_t) - \lambda^H r_3(s_t), & \text{otherwise.} \end{cases} \tag{10}$$

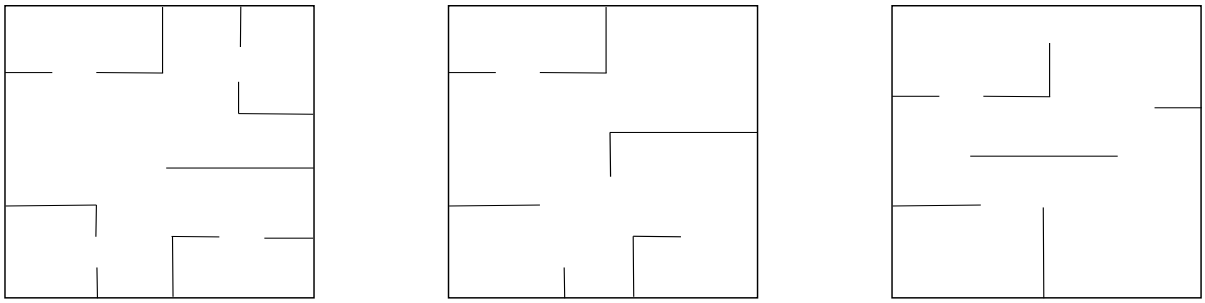
where λ^g, λ^o and λ^H are scalar weighting factors for the three reward terms $r_1(s_t)$, $r_2(s_t)$ and $r_3(s_t)$.

Reinforcement Learning algorithm and Neural Networks Architecture

DDPG is chosen as the candidate RL algorithm for determining the velocity commands (linear and angular velocities) of the robot in order to navigate to target locations without colliding with obstacles. It should be pointed out that the RL algorithm represents the high-level controller of the robot. Once the robot's navigation actions are determined, the robot's low-level controller executes each action by sending the appropriate torque commands to each actuator. To achieve smooth trajectories, the action space is continuous. In particular, the linear velocity is a continuous function limited in the range [0,1] to allow only forward motion (no backward) and the angular velocity is a continuous function limited in the range [-1,1] to allow right and left rotations. The state vector s_t is composed by 100 LiDAR data points of the 360 degrees range z_t, the action chosen at the previous time step a_{t-1} and the current distance from the target position expressed in Euclidean coordinates g. The state vector has dimension 104 and it is shown in Equation (11).

$$s_t = [z_t, a_{t-1}, g] \tag{11}$$

The network architecture consists of mainly three parts: feature extraction, policy (actor) and critic networks as shown in Figure 2. The first part of the network is responsible for extracting features from the laser range finder data through 1D-convolutional layers. Three 1D-convolutional layers with 8 filters each, stride length 2, ReLU activations and kernel sizes of 5, 3 and 3 respectively are used to extract



(a) *Env*-1. (b) *Env*-2. (c) *Env*-3.

Figure 3: The robot is trained on *Env*-1, in Figure 3a. Then, the performance of the policies are evaluated on the unseen a priori *Env*-2 and *Env*-3. The environments are all of size $4.5\,\mathrm{m}$ x $4\,\mathrm{m}$.

high-level features. These convolutional layers are used to achieve better generalization in unseen environments. The second part of the network is the policy network. This is composed of three fully-connected layers responsible of estimating the optimal linear and angular velocities of the robot based on the extracted features from the laser data along with the robot's speed in the previous time frame and target position in the robot's local frame. To constrain the range of linear velocity in [0,1] and angular velocity in [-1,1], as mentioned beforehand, sigmoid and tanh activation functions are used respectively in the output layer. However, the remaining layers employ ReLUs. Finally, the critic network estimates the Q-values of the state and action pairs using two fully-connected layers respectively with ReLu and linear activations.

Experimental design

Simulation Setup

The virtual 3D environment is built using the Robot Operating System (ROS) and Gazebo simulator. The experiments were conducted on an Ubuntu 16.04 machine with an Intel Core i7-8550 CPU. The algorithms are written using OpenAI package provided by the ROS middleware. The simulated platform is a skid-steering Husarion mobile robot. The robot is controlled through velocity commands (linear and angular velocities) that are directly sent to the low-level controller, with a frequency equal to 10 Hz, where the control loop waits until the command gets executed. This feedback is provided by estimating the robot's velocity from the encoder's readings. Furthermore, the robot is equipped with 360 degrees 2D laser range scanner (LiDAR) for sensing the environment.

Configuration of the RL and SLAM algorithms

For training the model, stochastic policy gradient with Adam optimizer (Kingma and Ba 2015) is employed to train both the actor and critic networks. However, for the actor network, a learning rate of 10^{-4} is used whereas the critic is updated using a learning rate of 10^{-3}. Furthermore, L2-regularization is included with a coefficient of 10^{-2} when training the critic network to prevent overfitting. A discount factor of $\gamma = 0.99$ and target update, $\tau = 0.001$ is used. The hyperparameters are selected based on the ones used in the original paper for the DDPG (Lillicrap et al. 2016). The exploration noise is chosen as an Ornstein-Uhlenbeck process with parameters $\sigma = 0.2$ and $\theta = 0.15$, since these values have empirically shown good performances.

In order to simultaneously map the environment and estimate the robot pose, the ROS Gmapping SLAM package is used (Grisetti, Stachniss, and Burgard 2007). A probability value is assigned to each cell based on whether it is occupied or free according to the laser sensor and odometry readings. Similar to (Grisetti, Stachniss, and Burgard 2005), the occupancy threshold value is chosen equal to 0.65 which means that if the probability value of the cell is greater than this value, this cell is occupied and, consequently, free otherwise. Based on the probability assigned to every grid cell, the entropy of the map is calculated according to equation (9). The complete list of parameters' values used in the experiments can be found in Table 1.

parameter	value
optimizer	ADAM
actor learning rate	10^{-3}
critic learning rate	10^{-4}
L2-regularization coefficient	10^{-2}
discount factor γ	0.99
target networks update τ	0.001
OU-noise σ	0.2
OU-noise θ	0.15
batch size	64
occupancy threshold	0.65
map update threshold	1.0
grid cell size	$5\,\mathrm{cm} \times 5\,\mathrm{cm}$
LiDAR max. range	$3\,\mathrm{m}$

Table 1: Parameters of the experiments.

Training and Evaluation

To validate the effectiveness of the proposed approach, we compare the performances of the path planner trained using the proposed reward function, in Equation (10), with the performances of the path planner trained used a TD reward function, as in (Tai, Paolo, and Liu 2017), and one trained

with a TD and OA reward function, as in (Zhang, Zhang, and Liu 2018). For a fair comparison, the same RL parameters are used as well as the same neural network architectures. The three different RL-agents are trained in *Env*-1, shown in Figure 3a, and then tested on the same set of 100 randomly generated targets environment *Env*-1. For each training episode, the target location g and the initial pose of the robot $\mathbf{p}_0$ are sampled from a uniform distribution to avoid biasing the policy toward specific targets and environment topology. For the path planner trained using (10), the map is also reset at the start of a new episode and only used when training the agent.

Furthermore, to assess the generalization properties of the policies learned in *Env-1*, we evaluate the planners on the same sets of 100 randomly generated targets in a priori unseen environments without further retraining: *Env*-2 and *Env*-3, in Figure 3b and 3c respectively. Moreover, we compare the RL-agents with move_base, the DWA path planner (Fox, Burgard, and Thrun 1997) of the navigation stack in ROS. Differently from the proposed approaches move_base requires the complete map of the environment to be known beforehand.

Results and discussion

Training Results

To compare the training performances of the RL-agents, we analyze the collision ratio in relation to the episode number. The results are shown in Figure 4.

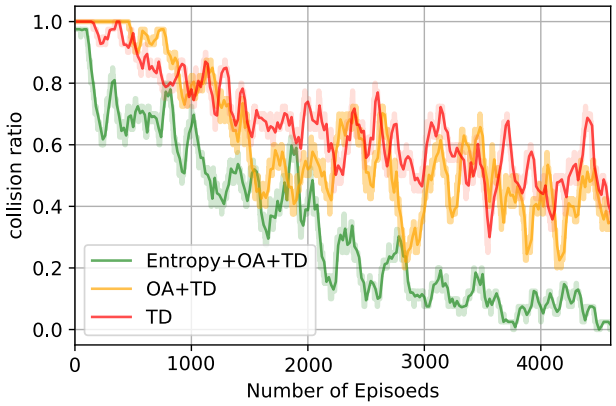


Figure 4: Evolution of the collision ratio with the number of training episodes. The collision samples of the proposed approach (green) decrease much faster than the TD only (red) and TD and OA (yellow) reward functions.

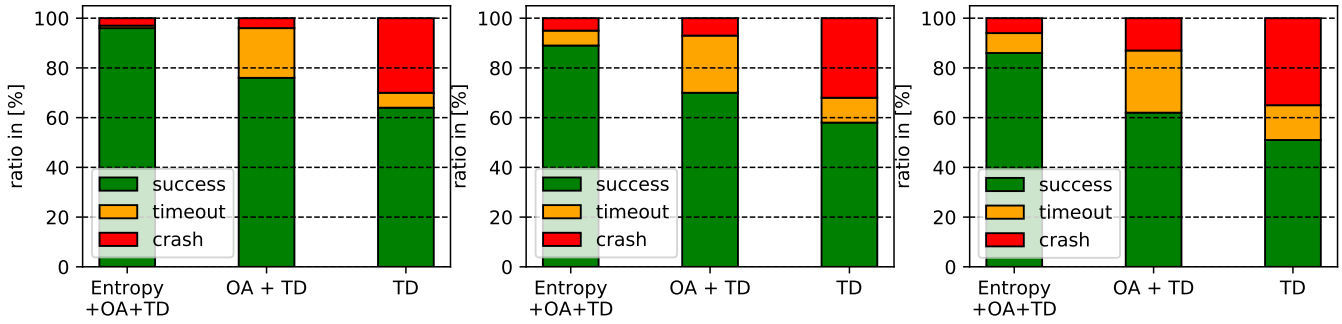
The agent trained with the TD reward function only (red line in Figure 4) struggles to reduce the collisions during training as it receives a penalty only when the collision has already happened. Learning to navigate safely and far away from the obstacle is more challenging when only sparse rewards are used. When the environment is fairly complex, the agent trained with TD and OA reward function (yellow line in Figure 4), even though slightly superior, still suffers from the same problems of the TD only. Eventually, the agent trained with (10) achieves the best training performances by achieving a collision ratio constantly smaller than 0.2 in only 3000 episodes. This is because the entropy term pushes the agent to better explore the environment, reach open areas and consequently escape local minima. Furthermore, throughout the training, the agent trained with the proposed approach is the only one able to consistently reduce the fluctuation in the collision ratio by showing higher robustness and generalization skills. These fluctuations are due to the random spawning of the robot and the target location at the beginning of each episode.

Evaluation Results

After training the three RL-agents in *Env*-1, the learned policies are evaluated on the same set of 100 randomly generated targets. To guarantee fair comparison the percentage of the successes, crashes, timeouts (episodes ended without either reaching the target or colliding with an obstacle) and the number of actions takes is recorded and shown in Figure 5 and Table 2.

In the training *Env*-1, the planner trained with TD reward function (7) can reach only 64% of the targets and, in most of the failed episodes, it collides with an obstacle. The planner trained with TD and OA reward function (8) achieves better performances as it can reach 76% of the targets and reduces the crashes compared to TD only (from 32% to 3%). However, the enhanced obstacle awareness, when the environment is complex and presents local minina, prevents the agent to collide, but it is not enough to prevent to get stuck in a room until the maximum number of action is reached. This can be noticed in the increment of the timeouts with respect to TD only (from 4% to 21%). On the other hand, the planner trained with the reward function (10) is the only one that can achieve good performances as it reaches the target in 96% of the episodes.

The crucial aspect for any learning-based approach is the generalization to untrained situations. To test this, we transfer the policies learned on *Env*-1 to the unseen *Env*-2 and *Env*-3 environments. Consistently with the training results, the proposed approach outperforms the two others in terms of success ratio, reduction of the collisions and timeouts as summarized in Table 2. For all the three environments we record and analyse the average number of steps per episode and the standard deviation as important elements for assessing length and smoothness of the trajectories of the different RL-agents. These results are presented in Table 2 as well. The agent trained with TD reward function (7) learns an overcautious behaviors and travels far away from the obstacles. This penalizes the trajectory length as it can be seen by the high average number of steps. When trained with TD and OA reward function (8), the agents can reduce the cautiousness and learn shorter paths in terms of average number of steps. However, when local minima are present, as in *Env*-2, the planner gets stuck in them and its performances are not much better than TD. The planner trained with the reward function (10) is able to learn shorter trajectories with respect to the other two thanks to the enhanced exploration granted by the entropy term. Even in unseen environment, the agent can escape local minima, thus it has effectively incorporated



(a) *Env*-1. (b) *Env*-2. (c) *Env*-3.

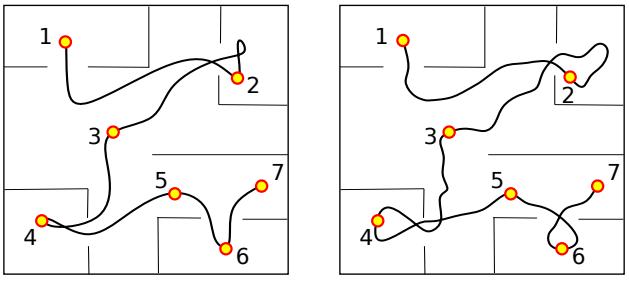
Figure 5: Performance comparison in the different environments of the three RL-planners.

and integrated the navigation skill with the exploration one.

Table 2: Performance assessment of the planners in *Env*-1, *Env*-2 and *Env*-3.

	approach	success ratio %	number of actions (mean ± std)
Env-1	TD	64 %	113.07±85.43
	OA+TD	76%	84.56 ± 76.31
	Entropy+OA+TD	**96%**	**59.28±37.99**
Env-2	TD	58 %	140.68±99.17
	OA+TD	68%	131.34± 86.78
	Entropy+OA+TD	**89%**	**124.07±74.86**
Env-3	TD	51 %	115.55±84.38
	OA+TD	62%	76.32±67.48
	Entropy+OA+TD	**84%**	**57.14±40.39**

Eventually, we compare the trajectories generated by our RL path planner with the one generated by move_base on the same set of targets (see Figure 6) that has to be reached in sequence in *Env*-1. Both planners can successfully reach all targets, however, the total travelled distance required to complete the whole path by the proposed planner is shorter; $14.7\,\mathrm{m}$ compared to $16.3\,\mathrm{m}$ required by the move_base. In addition, the path generated by move_base does not seem to be as smooth as the one by the proposed planner.



(a) proposed planner (b) move_base

Figure 6: A comparison between baseline motion planner and our proposed map-less motion planner.

Conclusions

The paper presents a DRL path planner for navigation in unknown environments. The planner is trained by exploiting the knowledge stored in the occupancy grid map, built online using Rao-Blackwellized particle filters (SLAM). However, the planner doesn't rely on any map except that during training. In particular, we use the map entropy to shape the reward function to improve the exploration skill of the planner to escape local minima (very common in complex environments). By learning better exploration skills, compared to a target driven (TD) reward function and obstacle aware (TD+OA) reward function, the agent trained with the proposed approach outperforms the other two not only in the training environment (success ratio of 96% against 64% and 76% respectively) but also in two different unseen environments (success ratio of 89% and 84% against 58%, 51% and 68%, 62% respectively). Furthermore, the agent trained with the proposed approach achieves performances close to the ones of move_base, DWA planner that requires the map of the environment.

References

[Botteghi et al. 2020] Botteghi, N.; Sirmacek, B.; Mustafa, K.; Poel, M.; and Stramigioli, S. 2020. On reward shaping for mobile robot navigation: A reinforcement learning and slam based approach. *arXiv:200204109*.

[Brunner et al. 2017] Brunner, G.; Richter, O.; Wang, Y.; and Wattenhofer, R. 2017. Teaching a machine to read maps with deep reinforcement learning. *arXiv:171107479*.

[Duo et al. 2019] Duo, N.; Wang, Q.; Lv, Q.; Wei, H.; and Zhang, P. 2019. A deep reinforcement learning based mapless navigation algorithm using continuous actions. In *2019 International Conference on Robots Intelligent System (ICRIS)*, 63–68.

[Fox, Burgard, and Thrun 1997] Fox, D.; Burgard, W.; and Thrun, S. 1997. The dynamic window approach to collision avoidance. *Robotics and Automation Magazine, IEEE* 4:23–33.

[Grisetti, Stachniss, and Burgard 2005] Grisetti, G.; Stachniss, C.; and Burgard, W. 2005. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals

and selective resampling. *Proceedings of the 2005 IEEE international conference on robotics and automation.*

[Grisetti, Stachniss, and Burgard 2007] Grisetti, G.; Stachniss, C.; and Burgard, W. 2007. Improved techniques for grid mapping with rao-blackwellized particle filters. *Robotics, IEEE Transactions on* 23:34 – 46.

[Kingma and Ba 2015] Kingma, D., and Ba, J. 2015. Adam: a Method for Stochastic Optimization,. In *International Conference on Learning Representations*, 1–15.

[Lillicrap et al. 2016] Lillicrap, P.; Hunt, J.; Pritzel, A.; Heess, N.; Erez, T.; Tasaa, Y.; Silver, D.; and Wierstra, D. 2016. Continous control with deep reinforcement learning. *International Conference on Learning Representations*.

[Mnih et al. 2013] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv:13125602*.

[Moravec 1988] Moravec, H. 1988. Sensor fusion in certainty grids for mobile robots. *AI Magazine* 61–74.

[Murphy 1999] Murphy, K. 1999. Bayesian map learning in dynamic environments. *Neural Information Processing Systems* 12:1015–1021.

[Mustafa et al. 2019] Mustafa, K.; Botteghi, N.; Sirmacek, B.; Poel, M.; and Stramigioli, S. 2019. Towards continuous control for mobile robot navigation: A reinforcement learning and slam based approach. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2/W13:857–863.

[Pfeiffer et al. 2017] Pfeiffer, M.; Schaeuble, M.; Nieto, J.; Siegwart, R.; and Cadena, C. 2017. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. *2017 IEEE International Conference on Robotics and Automation (ICRA)*.

[Stentz 1994] Stentz, A. 1994. Optimal and efficient path planning for partially-known environments. *IEEE International Conference on Robotics and Automation* 3310–3317.

[Sutton and Barto 1998] Sutton, R. S., and Barto, A. 1998. *Introduction to reinforcement learning*.

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

[Thrun, Burgard, and Fox 2005] Thrun, S.; Burgard, W.; and Fox, D. 2005. Probabilistic robotics. *MIT Press*.

[Zhang et al. 2016] Zhang, J.; Springenberg, J.; Boedecker, J.; and Burgard, W. 2016. Deep reinforcement learning with successor features for navigation across similar environments. *arXiv:161205533*.

[Zhang et al. 2017] Zhang, J.; Tai, L.; Boedecker, J.; and Liu, M. 2017. Neural slam: Learning to explore with external memory. *arXiv:170609520*.

[Zhang, Zhang, and Liu 2018] Zhang, W.; Zhang, Y.; and Liu, N. 2018. Danger-aware adaptive composition of drl agents for self-navigation. *arXiv:180903847*.

[Zhelo et al. 2018] Zhelo, O.; Zhang, J.; Tai, L.; Liu, M.; and Burgard, W. 2018. Curiosity-driven exploration for mapless navigation with deep reinforcement learning. *arXiv:180400456*.