TOWARDS CONTINUOUS CONTROL FOR MOBILE ROBOT NAVIGATION: A REINFORCEMENT LEARNING AND SLAM BASED APPROACH

Khaled A. A. Mustafa[1*], Nicolò Botteghi[1*], Beril Sirmacek[1*], Mannes Poel[2], Stefano Stramigioli [1]

[1] Robotics and Mechatronics, Faculty of Electrical Engineering, Mathematics and Computer Science,
University of Twente, The Netherlands (e-mail:k.a.a.mustafa@student.utwente.nl, {n.botteghi, b.sirmacek, s.stramigioli}@utwente.nl)

[2] Data Science, Faculty of Electric Engineering, Mathematics and Computer Science,
University of Twente, The Netherlands (e-mail: m.poel@utwente.nl)

KEY WORDS: SLAM, Deep Reinforcement Learning, Artificial Intelligence, Online Path Planning, Unknown environments

ABSTRACT:

We introduce a new autonomous path planning algorithm for mobile robots for reaching target locations in an unknown environment where the robot relies on its on-board sensors. In particular, we describe the design and evaluation of a deep reinforcement learning motion planner with continuous linear and angular velocities to navigate to a desired target location based on deep deterministic policy gradient (DDPG). Additionally, the algorithm is enhanced by making use of the available knowledge of the environment provided by a grid-based SLAM with Rao-Blackwellized particle filter algorithm in order to shape the reward function in an attempt to improve the convergence rate, escape local optima and reduce the number of collisions with the obstacles. A comparison is made between a reward function shaped based on the map provided by the SLAM algorithm and a reward function when no knowledge of the map is available. Results show that the required learning time has been decreased in terms of number of episodes required to converge, which is 560 episodes compared to 1450 episodes in the standard RL algorithm, after adopting the proposed approach and the number of obstacle collision is reduced as well with a success ratio of 83% compared to 56% in the standard RL algorithm. The results are validated in a simulated experiment on a skid-steering mobile robot.

1. INTRODUCTION

Autonomous navigation of robots in unknown environments from their current position to a desired target location without colliding with obstacles represents an important aspect in the field of mobile robots. In literature, traditional methods do exist in case a complete knowledge of the environment is available including cell decomposition and potential filed approaches. However, this is not the case in real-life applications where a complete knowledge about the environments can be hardly obtained due to its intrinsic stochasticity.

The challenge of navigation in unstructured environments can be formulated as a reinforcement learning (RL) problem where the agent learns the optimal path through a straightforward trial and error process by interacting with the environment. During the interaction, the agent perceives the environment through its sensors and affects the environment through actions performed by its actuators. By applying an action, the agent is able to change its own state and the state of the environment and consequently it receives a reward or a penalty for being in that state. The reward is the way of teaching to the agent whether the action taken in that state is good or bad. Accordingly, the optimal action for each state can be discovered by maximizing a predefined accumulated reward that reflects the quality of the trajectory taken by the robot. In (Zhang et al., 2017a), a successor feature DQN based reinforcement learning is proposed to solve the navigation problem when a map of the environment is known a priori. The main focus was to transfer the knowledge from one environment to another where the input is depth images obtained through a kinetic sensor and the output is four discrete actions for robot's navigation. In (Brunner et al., 2018), Asynchronous Advantage Actor-Critic (A3C) approach was proposed to help a robot moving out of a random maze for which a map is given. The input to the system includes 2D map of the environment, the robot's heading and the previous estimated pose whereas the output is the navigation actions such as move (forward, backward, right and left). Furthermore, in (Zhang et al., 2017b), an external memory acting as an internal representation of the environment for the agent is fed as an input to a deep reinforcement learning algorithm. In this way, the agent is guided to make informative planning decisions to effectively explore new environments. The work presented in this paper is built upon the algorithm proposed in (Tai et al., 2017) where a mapless navigation is proposed based on an asynchronous deep deterministic policy gradient algorithm.

The purpose of this paper is twofold. On one hand, it is aimed to enhance the navigation capabilities to navigate a skid-steering mobile robot (SSMR) with non-holonomic constraints in an unknown environment without collisions and with the least amount of feasible actions. This goal is achieved by implementing a deep deterministic policy gradient (DDPG) through an off-policy actor-critic algorithm where the input is sparse laser data extracted from a laser range finder and the output is continuous navigation actions. The main advantage of utilizing DDPG algorithm is due to the fact that following policy gradient to solve reinforcement learning tasks only slightly modifies the parameters of the policy in contrast to value based methods where large jumps between estimated policies are possible (Deisenroth et al., 2013). On the other hand, Simultaneous Localization and Mapping, also known as SLAM, technique is integrated with reinforcement learning in an attempt to improve the learning rate by defining a reward function based on the (partial) knowledge of the map and by providing more accurate estimation of the robots states. In addition, a comparison is made, in terms of the learning performance, in case a partial map is available to the robot as an output from a SLAM algorithm and when no map is available at all.

The rest of the paper is organized as follows. In section 2, the

*Those authors contributed equally

theory behind DDPG and Rao-Blackwellized particle filter is discussed. In section 3, the motion planner design is presented. Section 4 describes the experiments performed. Furthermore, sections 5 and 6 carry respectively the discussion of the results and the conclusions.

2. THEORETICAL BACKGROUND

2.1 Deep Deterministic Policy Gradient

The actor-critic framework, deep deterministic policy gradient (DDPG) provides an improvement to deep Q-networks (DQN) by making it tractable for continuous action spaces. Both the actor and the critic are described by two separate neural networks. The current state is given as input to the actor network which outputs a single real value representing an action chosen from a continuous action space $\mathcal{A}$. On the other hand, the critics output is the estimated Q-value based on the current state and the action given by the actor. The deterministic policy gradient theorem provides the update rule for the weights of the actor network whereas, the critic network is updated from the gradients obtained from the TD error signal.

2.1.1 Actor Network: In order to update the parameters of the actor neural network, the following equation is applied:

$$\theta^{\pi}_{t+1} = \theta^{\pi}_{t} + \alpha \nabla_{\theta^{\pi}} J(\pi_\theta), \quad J(\pi_\theta) = \mathbb{E}_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_k\right) \tag{1}$$

where $\mathbb{E}_{\pi}[.]$ denotes the expected value with respect to the policy π. The main advantage of following policy gradient to solve reinforcement learning tasks is that it slightly modifies the parameters of the policy which guarantees smooth transition between states (Deisenroth et al., 2013). As shown in equation (1), it is required to evaluate the gradient of the expected return $\nabla_{\theta^{\pi}} J(\pi_\theta)$. The policy gradient theorem (Sutton et al., 1999) states that the corresponding gradient of the return, using likelihood-ratio trick, in case of stochastic policy can be given as

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \int_{\mathcal{S}} \rho^{\pi}(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^{\pi}(s,a) \mathrm{d}a \mathrm{d}s \\ &= \mathbb{E}_{s\sim\rho^{\pi}, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta\left(a|s\right) Q^{\pi}(s,a)\right] \end{aligned} \tag{2}$$

The above equation shows that the gradient is an expectation of both states and actions. Therefore, large number of samples from both action and state space is required, in principle, in order to evaluate a good estimate of the gradient. However, by utilizing a deterministic policy instead, the mapping from state space to action space becomes fixed and accordingly there is no need to integrate over the whole action space. (Silver et al., 2014) introduces deterministic policy gradient algorithm that defines the gradient of the expected return subjected to deterministic policy. For the deterministic case, the gradient is given as

$$\begin{aligned} \nabla_{\theta^{\pi}} J(\pi_\theta) &= \int_{\mathcal{S}} \rho^{\pi}(s) \nabla_{\theta^{\pi}} \pi\left(s|\theta^{\pi}\right) \nabla_a Q\left(s, a|\theta^{Q}\right) \mathrm{d}s \\ &= \mathbb{E}_{s\sim\rho_{\pi}}\left[\nabla_{\theta^{\pi}} \pi\left(s|\theta^{\pi}\right) \nabla_a Q\left(s, a|\theta^{Q}\right)|_{a=\pi(s|\theta^{\pi})}\right] \end{aligned} \tag{3}$$

where $\rho_\pi(s)$ is the state distribution under policy π, θ^{π} is the parameter vector for the policy π and θ^{Q} is the parameter vector for Q-function (critic network). The deterministic policy gradients can be computed more efficiently than the stochastic case and these algorithms show significantly better performance than their stochastic counterpart (Silver et al., 2014) since they require fewer data samples to converge. As depicted in equation (3), it is required to get the gradient of the action-value function $Q\left(s, a|\theta^{Q}\right)$. For this reason, an estimate of it can be evaluated through a critic network that is discussed in the next subsection.

2.1.2 Critic Network: Q-learning is one of the prominent reinforcement learning algorithm that can be considered as a variant of temporal difference (TD) algorithm. In the simplest case, Q-learning algorithms employ a table to store each state-action pair. However, that makes these algorithms applicable only to environments with small number of states and actions. Typically, the problem is not only related to the amount of memory required to store the table, but also the time required to estimate each state-action pair accurately. For this reason, non-linear function approximators, e.g. (deep) artificial neural networks (ANN), are introduced in order to generalize Q-learning algorithms to larger state-action space which is called deep Q-learning (DQN). However, it was shown that utilizing neural networks as function approximators directly to Q-learning algorithms without further modifications leads to an unstable behavior and the convergence is no longer guaranteed (van Hasselt et al., 2016) . The main cause of this issue is that when using neural networks for reinforcement learning, it is assumed that the samples are independently distributed. However, this is not the case when the samples are generated sequentially since they are temporally correlated which results in high variance in the estimation. In order to tackle this problem, an *experience replay* is used to break the temporal correlation between the consecutive transitions where the agent's experience at each time step $e_t = \langle s_t, a_t, r_t, s_{t+1}\rangle$ is stored in a replay buffer $D_t = \{e_1, ..., e_t\}$. At each time step, a fixed number of samples, a mini-batch, is extracted randomly from the replay buffer and used to train the network. When the replay buffer is full, the oldest samples were discarded. This way, gradient descent methods from the supervised learning literature can be safely used, to minimize the TD-error squared.
The learning of the value-function in deep reinforcement learning is based on the adjustment of the neural network weights by minimizing the loss function, which corresponds to the mean squared error between the TD target and the current value function

$$L\left(\theta^{Q}\right) = \mathbb{E}_{s\sim\rho_{\pi(.)}, a\sim\pi(.)}\left[\left(\underbrace{Q\left(s_t, a_t|\theta^{Q}\right) - y_t}_{\text{TD error}}\right)^2\right] \tag{4}$$

where $y_t = r(s_t, a_t) + \gamma Q\left(s_{t+1}, a_{t+1}|\theta^{Q}\right)$. To minimize the loss function, the gradient of the loss function is computed with respect to the weights. In (Mnih et al., 2015), it is shown that implementing equation (4) directly results in divergence in many cases. The reason is that the updated $Q(s, a|\theta^{Q})$ is also used with the same weights in calculating the TD target y_t which makes the optimization appears to be chasing its own tail which introduces instability. One possible solution is to introduce a second network called target neural network that is proposed in (Mnih et al., 2013) to calculate target Q-values $Q'(s, a|\theta^{Q'})$ where the target network parameters θ' are only updated with the Q-network parameters θ every certain number of steps. The target-network and the Q-network share the same network architecture, but only the weights of the Q-network are learned and updated. Here it should be pointed out that concepts of experience replay and target network are also applied to the actor network. For target actor and critic networks, the parameter $\theta^{Q'}$ and $\theta^{\pi'}$ are updated respec-

tively by:

$$\theta^{Q'} = \tau\theta^{Q} + (1-\tau)\theta^{Q'}$$
$$\theta^{\pi'} = \tau\theta^{\pi} + (1-\tau)\theta^{\pi'} \quad (5)$$

The employment of target networks converts the problem of learning the optimal Q-function into a supervised learning problem which improves the stability of the algorithm immensely.

2.2 Grid-Based SLAM with Rao-Blackwellized Particle Filter

SLAM depicts the process of a robot creating a map of an environment while simultaneously estimating its location within the self-created map (Thrun et al., 2005). Currently, numerous SLAM approaches have been suggested and applied to a multitude of applications using different sensors, different algorithmic steps and different platforms.

In this section, a grid-based SLAM with Rao-Blackwellized particle filters introduced in (Murphy, 1999) is briefly discussed in order to provide an accurate estimate of the robot's pose and a partial map of the environment that can be utilized in the reward shaping of the reinforcement learning in an attempt to speed-up the learning rate.

The key idea of Rao-Blackwellized particle filter (RBPF) is to estimate the joint posterior $p\left(x_{1:t}, m|z_{1:t}, u_{1:t-1}\right)$ about the trajectory of the robot, that is the sequence of its poses, $x_{1:t} = x_1, ..., x_t$ and the map m given the odometry measurements $u_{1:t-1} = u_1, ..., u_{t-1}$ and a set of observations $z_t = z_1, ..., z_t$ obtained by the mobile robot. Thus, it incrementally processes the sensor observations and the odometry readings as they are available. The Rao-Blackwellized particle filter makes use of the following factorization of the conditional probability

$$p(x_{1:t}, m|z_{1:t}, u_{1:t-1}) = p(m|x_{1:t}, z_{1:t}).p(x_{1:t}|z_{1:t}, u_{t-1}) \quad (6)$$

The advantage of this factorization is the fact that it is possible to estimate the trajectory of the robot firstly using a particle filter where every particle represents a potential trajectory of the robot and then use this trajectory in order to estimate the map using mapping with known poses (Moravec, 1988) since $z_{1:t}$ and $x_{1:t}$ are known from the previous step.

The intuition behind particle filters is to approximate the belief of the robot's pose bel(x_t) by a set of particles $\mathcal{X}_t$. The next generation of particles $\mathcal{X}_{t+1}$ can be obtained from the previous generation $\mathcal{X}_t$ by sampling from a probabilistic odometry motion model $p(x_{t+1}^{(i)}|x_t^{(i)}, u_t)$ where i represents the i^{th} particle in the particle set $\mathcal{X}$. Then, by integrating the probabilistic observation model $p(z_{t+1}|x_{t+1}^{(i)})$, an individual importance weight $w_{t+1}^{(i)}$ is assigned to every particle. After that, particles are drawn with replacement proportional to their assigned importance weight. This step is called a resampling step. As a matter of fact, after resampling, all particles have the same weight. By incorporating the importance weights into the resampling process, the distribution of the particles changes where particle with low importance weights are depleted. In (Grisetti et al., 2007), an adaptive resampling technique is proposed in order to reduce the risk of particle depletion while learning an accurate map. For each particle, the corresponding map estimate $p(m^{(i)}|x_t^{(i)}, z_{1:t})$ is computed based on the estimated trajectory $x_{1:t}^{(i)}$ from the output of the particle filter and the history of observations $z_{1:t}$. The map considered in this paper is an occupancy grid map.

Occupancy grid maps address the problem of generating consistent maps from noisy and uncertain measurement data, under the assumption that the robot pose is known (Thrun et al., 2005). The occupancy grid map divides the worksapce into evenly spaced cells where a probability distribution is assigned to each cell in the grid indicating whether it is occupied or free. If the cell is not in the range of the sensors, it is considered as unknown. The posterior over maps given the trajectory of the robot $x_{1:t}$ and all the observations $z_{1:t}$ up to time t is given as $p(m|z_{1:t}, x_{1:t})$ where the controls $u_{1:t}$ play no role since the path of the robot is already known. The certainty of the estimation of the entire map can be broken down into the problem of estimating the posterior of every grid cell m_i in the map and then the posterior over the entire map can be approximately estimated by:

$$p(m|z_{1:t}, x_{1:t}) = \prod_{i=0}^{M} p(m_i|z_{1:t}, x_{1:t}) \quad (7)$$

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

3. METHODOLOGY

This section presents in details how the information extracted from the map can be utilized in the reward shaping of the DDPG algorithm in order to improve the convergence rate.

3.1 Motion Planner Implementation

We propose an end to end learning strategy, for which a robot learns to navigate through an unknown environment towards its desired goal by using only raw 10-dimensional sensory data from a laser range finder. The algorithm integrates SLAM with DDPG off-policy actor-critic algorithm to improve the performance of the agent's learning. As mentioned previously, the aim of this paper is to find the optimal path from the starting point of a skid-steering mobile robot to the target through DDPG algorithm proposed in (Lillicrap et al., 2016). Then, to integrate the (partial) knowledge of the map, obtained by the SLAM algorithmm, in the reward function to assess how much it speeds up the convergence rate. To achieve the first purpose, two neural networks are constructed to represent the the actor and the critic respectively. The actor network represents the policy and thus it is responsible for mapping states into actions $a_t = \pi(s_t)$. For the navigation problem, the states are selected to be the observation from the laser range finder that can be represented as 10-dimensional laser beams with 180° field of view (FOV) x_t, the relative distance between the target and the agent represented in polar coordinates p_t and the last action executed by the agent v_{t-1}.

$$v_t = \pi(s_t) = \pi(x_t, p_t, v_{t-1}) \quad (8)$$

The actor's neural network is composed of three fully-connected hidden layers with 512 nodes each which are activated by a rectified linear unit (ReLU) activation function. The output of the actor's network is a 2-dimensional vector representing the linear and angular velocities of the robot respectively. For this purpose, a sigmoid activation function is used to constrain the linear motion of the robot in the range between [0,1]. The reason why the backward motion of the robot is restricted is due to the chattering behavior of the robot observed at the early experiments due to the stochasticity of the behavioral policy. To constrain the angular velocity of the robot in (-1,1), a hyperbolic tangent function (tanh) is employed. The layout of the actor neural network is depicted in Figure 1. In addition, the output of the actor network

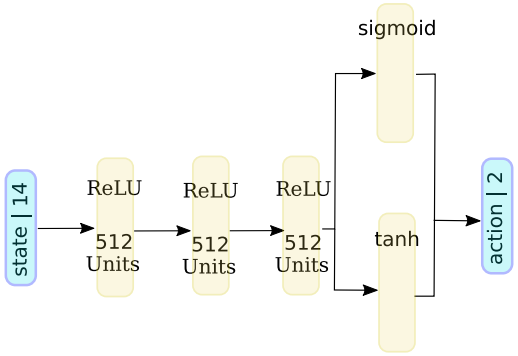

Figure 1. Layout of the actor network for DDPG. The networks has three fully connected layers with 512 neurons each with ReLU activiation. The network receives a 14-dimensional state vector and outputs a 2-dimensional continuous actions vector. The linear velocity is constrained in [0,1] using a sigmoid while the angular velocity is constrained in [-1,1] using a hyberbolic tangent

is further multiplied by hyperparameters to limit the maximum linear velocity of the robot to $0.25\mathrm{m/s}$ and the maximum angular velocity to $1\mathrm{rad/s}$. The actor outputs, thus the actions, are then sent to the low-level controller to control the motion of the robot's actuators. The critic network estimates the Q-value of a state-action pair and thus it takes both the state and the action as inputs; however, the action input skips the first layer. The output of the critic is an estimation of the reward. Based on the output of the critic network, the weights of both the actor and critic are updated accordingly. Again, the hidden layers are activated by a ReLU function whereas the Q-value is activated by a linear activation. Figure 2 shows the architecture of the critic network.

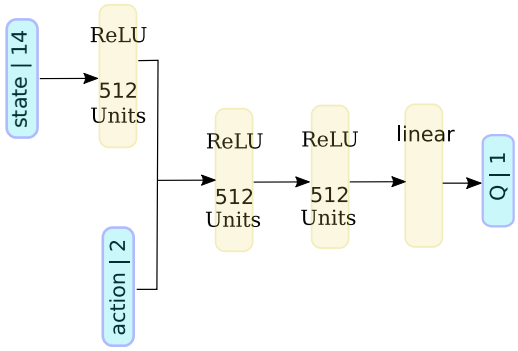

Figure 2. Layout of the critic network for DDPG. The networks has three fully connected layers with 512 neurons each with ReLU activation. The states are fed through the network from the first layer while the actions only from the second one. The output layer has linear activation function and outputs the estimated Q-value given the current state and the action taken.

The parameters of the actor and critic neural networks are updated according to,

$$\begin{aligned} \delta_t &= r_t + \gamma Q(s_{t+1}, \pi(s_{t+1})) - Q(s_t, a_t) \\ \theta^Q_{t+1} &= \theta^Q_t + \alpha_{\theta^Q} \delta_t \nabla_{\theta^Q} Q(s_t, a_t) \\ \theta^\pi_{t+1} &= \theta^\pi_t + \alpha_{\theta^\pi} \nabla_{\theta^\pi} \pi(s_t) \nabla_a Q(s_t, a_t)|_{a=\pi(s)} \end{aligned} \tag{9}$$

A major challenge of learning in continuous action spaces is exploration. Since DDPG algorithm uses a deterministic policy gradient in updating the weights of the policy network, an off-policy reinforcement learning algorithm is used where a stochastic behavioral policy is employed while the agent learns a deterministic one in order to guarantee adequate exploration of the environment. For this reason, an exploration policy $\pi'(s_t)$ is constructed by adding noise sampled from a noise process $\mathcal{N}$ to the output of the actor network,

$$\pi'(s_t) = \pi(s_t|\theta^\pi_t) + \mathcal{N} \tag{10}$$

In this work, the noise $\mathcal{N}$ is sampled from a temporally correlated Ornstein-Uhlenbeck (OU) process.
A 2D occupancy grid map of the surrounding is generated while the robot is exploring the unknown environment, using data extracted from laser range finder and the robot's odometry information. Every cell inside the occupancy grid is classified as (occupied, free, unknown) based on a predefined threshold value that determines the occupation probability of each cell. Furthermore, the occupation probability of every cell is being updated while the robot keeps exploring the environment.

3.2 Reward Shaping

Reward function is the most important aspect in reinforcement learning problem since the actions are selected in such a way that the cumulative reward is maximized. The reward signal is the mean by which the goal of the learning is specified for the agent. It is a designed application-specific function that, given the action of the agent and the state of the system, returns a single real number indicating how good or bad that action was. It corresponds to pleasure and pain in biological systems. Designing a good reward signal for a robotic reinforcement learning task can be challenging in different ways. This area of reinforcement learning, known as reward designing or shaping, is considered an art rather than a well-established science (Sutton and Barto, 2017). Reward functions can be a simple bonus when the agent reaches a target and, consequently, a penalty in case it hits an obstacle. This "sparse reward" is assigned to prioritize actions that make the agent reach the goal and penalize actions that make the agent colliding. On the other hand, it can be more sophisticated and depend on the distance between the agent and the target. This is called "dense reward".
In this work, two different reward functions are selected and a comparison of the performance of the agent based on each function is made.

3.2.1 Reward based on RL only: The reward function (11) doesn't integrate the available knowledge of the environment and it is formulated based on the exponential Euclidean distance between the agent and the target position. Moreover, a bonus is given in case the agent reaches the target r_{reached} with some predefined tolerance. This additional sparse reward term is necessary in particular when obstacles are located close to the target location to encourage the robot to navigate towards the goal even if it gets some negative immediate reward due to the fact that it gets close to these obstacles. Furthermore, if the robot gets too close to an obstacle, it would receive a high negative reward (penalty). Here it should be pointed out that the episode is terminated in three scenarios: i) the agent reaches the goal with some tolerance d_{min}, ii) the agent get closer to an obstacle with a minimum threshold, iii) the agent exceeds the maximum number of allowed time-steps T in every episode without either reaching the target or hitting an obstacle. The maximum number of iterations is a hyperparameter that is tuned based on the average number of actions required by the agent to reach the goal observed during the early experiments. The reward $r(s, a)$ is given after executing every navigation action a_t and can be, mathematically, formulated

as:

$$r(s_t, a_t) = \begin{cases} r_{\text{reached}}, & d < d_{min}. \\ r_{\text{crashed}}, & s_{ts} \\ 1 - e^{\gamma d}, & \text{otherwise} \end{cases} \tag{11}$$

where d is the euclidean distance between the agent and the target, γ is a hyper-parameter that can be tuned and s_{ts} represents an undesirable terminal state including getting too close to an obstacle or exceeding the maximum number of steps allowed in an episode.

3.2.2 Reward based on RL and SLAM: In this section, the reward function is shaped based on the available knowledge about the environment gained through the robot's experience. This knowledge is provided by a 2D occupancy grid map built by the SLAM algorithm discussed in section 2.2. In that sense, the reward function does not only depend on how far the agent is from the target but on the distance to the multi-obstacles inside the workspace as well. The incorporation of the environment's knowledge should be weighted by the level of certainty of the map's posterior. This can be formulated as follows:

$$r(s_t, a_t) = \begin{cases} r_{\text{reached}}, & d < d_{min} \\ r_{\text{crashed}}, & s_{ts} \\ 1 - e^{\gamma d} - \underbrace{p(m|z_{1:t}, x_{1:t}) \sum_{i=0}^{k} e^{-c_{min}}}_{\text{MAP-dependent term,}} & \text{otherwise} \end{cases} \tag{12}$$

where k is the total number of occupied cells in the vicinity of the robot, c_{min} is the distance between the robot and the occupied cell. Here it should be noted that the last term in equation (11) has only an effect on the reward when the robot approaches the occupied cell with a minimum threshold.

4. SIMULATED EXPERIMENTS

The main objective of the proposed algorithm is to find the shortest trajectory from the current location of the robot to the target with minimum executed actions. The actor-critic algorithm is experimented on a gazebo simulator representing the 3D environment. The experiments were conducted on an Ubuntu 16.04 machine with an Intel Core i7-8550 CPU and a NVIDIA Jetson TX2 GPU. The algorithm is implemented using OpenAI package provided by the Robot Operating System (ROS) middleware. The simulated environment contains cuboid objects representing the obstacles and a target for the agent to reach, rendered as red & white circle, as shown in Figure 3. The simulated platform is a Husarion mobile robot with skid-steering model. The actor and critic networks are initialized with two neural networks having three hidden layers with 512 hidden neurons that are activated by ReLU activation function, as described in section 3.1. For training the model, stochastic policy gradient with ADAM (Kingma and Lei Ba, 2015) optimizer is employed and the learning rates are taken to be 0.0001 and 0.001 for the actor and critic respectively. A discount factor of $\gamma = 0.99$ and target update, $\tau = 0.001$ is used. The initial weights and biases of the hidden neurons are chosen from a uniform distribution $[-\frac{1}{\sqrt{f}}, \frac{1}{\sqrt{f}}]$ where f is the number of inputs to the layer. The weights and biases for the output layer are taken from a uniform distribution $[-3 \times 10^{-3}, 3 \times 10^{-3}]$ to ensure that the outputs at the start

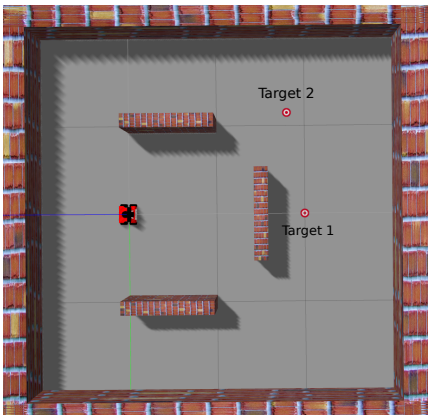

Figure 3. The virtual training environment simulated in gazebo. A Husarion robot is used as the platform

of training are close to zero. The exploration noise is modeled as an Ornstein-Uhlenbeck process with parameters, $\sigma = 0.2$ and $\theta = 0.15$. The outputs of the policy are clipped to lie between the actuator limits after the addition of noise. The maximum dimension of the replay buffer is selected to be 100000, which means that, in the worst case, the buffer can store 100 episodes since the maximum number of iterations in every episode is limited to 1000 time-steps. The update of the weights of the networks are executed with a mini-batch of dimension 64. It is worth mentioning that a small batch size could lead the algorithm to get stuck into specific portion of the environment (local minima) whereas large batch-size can make the training period much longer since the network would be trained for more data. Thus, a good trade-off of 64 is selected so as not to elongate the training period and to ensure training on larger areas of the environment. In this work, the robot is trained in a $4 \times 4\text{m}^2$ area with multiple obstacles. In order to simultaneously map the environment and estimate the robot pose, the ROS gmapping SLAM package was used. The inputs for mapping included wheel odometry and laser rangefinder data and a 2D occupancy grid map representing the environment is one of the outputs. The grid size of every cell is 1cm $\times$ 1cm resulting in 400×400 cells. A probability value is assigned to each cell based on whether it is occupied or free according to the laser sensor and odometry readings. An occupancy threshold is assigned a value of 0.65 which means if the probability value of the cell is greater than this value, this cell is occupied and, consequently, free otherwise. Since, a single obstacle is represented by multiple occupied cells,based on its size, it is better to prepossess this data by selecting only a certain number of grids in order to avoid iterating over redundant cells. Besides that, to avoid higher computational complexity of the calculations, the map is only updated after certain change occurs to the probability of the posterior of the map $p(m|z_{1:t}, x_{1:t})$ within a threshold of 0.2. The map-dependent term in equation (12) has an impact on the immediate reward when the minimum distance between the robot and the obstacle becomes smaller than 0.5m.

The robot subscribes to laser readings with a scanning range from 0.2m to 1.3m. The position of the robot is evaluated through Rao-Blackwellized particle filter, instead of using raw odometry data, in order to calculate the polar coordinates from the target position that is fed as an input to the policy network. The agent is free to select any angular and linear velocities from a continuous space as long as they are feasible by the physical constraints of the robot. These velocity commands are directly sent to the low-level controller where the algorithm waits until the command gets executed. After the termination of every episode, the environment

is reset and the robot returns back to its initial configuration. For the sake of comparison, in the following experiments, the agent is trained to reach different target locations with and without incorporating the map knowledge inside the reward function.

5. RESULTS

To evaluate the proposed algorithm, the results of different scenarios are discussed here. For every scenario, the performance of the algorithm is analyzed through the average cumulative reward that is given by

$$R' = \frac{\sum_{n=1}^{N} R(n)}{N}$$

where R' is the average cumulative reward at the end of the episode, N is the number of time-steps experienced by the agent before the episode is terminated and $R(n)$ is the reward at the considered time-step n. This analysis criterion is useful to verify if the agent follows an optimal policy since the main aim is to reach the desired target with the least amount of actions and without collisions. In the first scenario, the target (Target1) location lies 0.6m behind one of the obstacles as shown in Figure 3. The results show that incorporating map knowledge inside the reward function dramatically outperforms the standard DDPG algorithm as shown in Figure 4, where the mean value of the accumulated reward is calculated every 30 episodes. As depicted in Figure 4, the combined algorithm takes about 560 episodes before it converges to the optimal path compared to about 1450 episodes that are required by the standard algorithm. The reason is that, with a single sparse penalty on the collisions, the agent needs to collide with the obstacle in front many times in order to execute a turning maneuver as it can only realize that there is an obstacle after hitting it. On the other hand with the proposed reward function (12), since the reward is a dense function around the known obstacles, the agent realizes the optimal path much faster. Furthermore, the success ratio in case of the combined approach is approximately 83% compared to 56% that is achieved by the standard algorithm. This is another clear indication that the number of obstacles' collisions during training has decreased significantly as well. Here, it should be noted that, we compared the number of episodes required to converge to the optimal value rather than the value of the reward when the algorithm has converged, since the reward functions for both algorithms are different. This aspect can be noticed in Figure 4, where in the last 400 episodes the median value of the cumulative reward not dependent on the map is slightly higher than the cumulative reward defined based on the knowledge of the map. This is due to the fact that the goal is located next to an obstacle and the negative map-dependent penalty acts by reducing the total value reward.

In addition, to guarantee that the proposed approach is not target-based, the robot was trained to reach a different target, labelled as Target 2 in Figure 3. The results of this experiment are illustrated in Figure 5. Even in this case, the combined approach still gives better results. The difference is not as significant as it is in the first scenario, but this is reasonable since the maneuver, in this case, does not involve too much interaction with the surrounding obstacles. The combined approach converges after about 125 episodes whereas the standard one converges after nearly 520 episodes. It is also obvious that at the beginning of the training, the combined approach achieved quite low rewards because of the high uncertainties about the map. However, once the knowledge is obtained, the performance improves significantly which proves the importance of incorporating the online-acquired map knowledge in the reward shaping function.

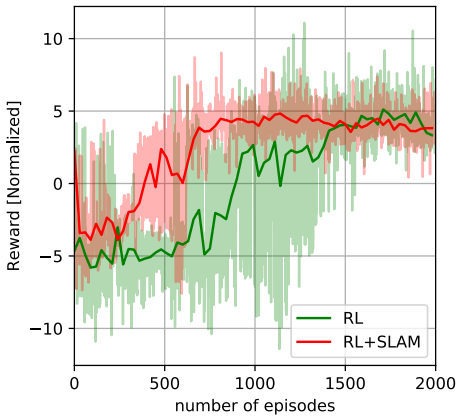

Figure 4. Normalized reward per episode for the trained agent (Target 1). The proposed reward function (red) guarantees drastically improved the convergence rate with respect to a reward function that doesn't use the knowledge of the map (green). Notice that because the reward functions are different, the median value of the reward is not taken into account when making the comparison.

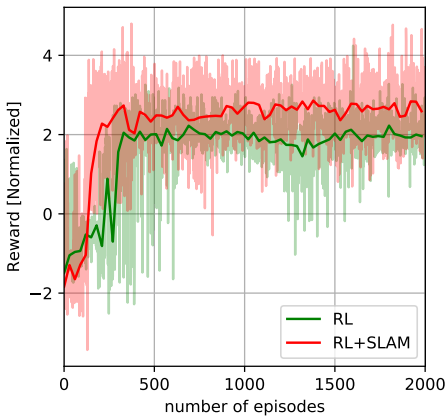

Figure 5. Normalized reward per episode for the trained agent (Target 2). Even when the path doesn't require go around an obstacle, the proposed reward function (red) outperform a reward function that doesn't use the knowledge of the map (green).

6. CONCLUSION

In this study, a reinforcement learning and SLAM-based combined approach is proposed for mobile robot's navigation in an unknown environment. The algorithm makes use of DDPG to obtain continuous velocities for the robot. A grid-based SLAM with Rao-Blackwellized particle filter algorithm was incorporated with the RL algorithm in order to improve the performance of the latter. The performances of the proposed algorithm were assessed based on a comparison with the most commonly used and efficient reward function for navigation tasks (11) in the state of art. It has been proven that shaping the RL reward function RL based on the knowledge of the map improves drastically the convergence rate, in terms of the number of episodes required to converge and decreases the number of collisions with obstacles. The key benefit of the proposed method lies in the possibility to receive negative rewards several steps before the effective collision with the obstacles even in cases of partial knowledge of the map. This aspect gives the possibility to the network to learn, in earlier stages, to avoid getting too close to obstacles. As a future work, the generalization properties of the proposed algorithm, during the testing phase (so without training the network parameters), needs to be evaluated in the case of different target and obstacles locations.

ACKNOWLEDGEMENT

Our experiments benefit from NVIDIA Jetson TX2 which is received as a research grant by Beril Sirmacek.

REFERENCES

Brunner, G., Richter, O., Wang, Y. and Wattenhofer, R., 2018. Teaching a machine to read maps with deep reinforcement learning. *The thirty second AAAI conference on Artificial Intelligence* pp. 2763–2770.

Deisenroth, M., Neumann, G. and Peters, J., 2013. A survey on policy search for robotics. *Foundations and Trends in Robotics* pp. 1–141.

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

Kingma, D. and Lei Ba, J., 2015. Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

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, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. *NIPS Deep Learning Workshop*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersin, S., Beatti, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D., 2015. Human-level control through deep reinforcement learning. *Nature International Journal of Science* pp. 529–533.

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

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

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. and Riedmiller, M., 2014. Deterministic policy gradient algorithms. *31st International Conference on Machine Learning* 32, pp. 1309–1332.

Sutton, R. and Barto, A., 2017. Reinforcement learning: An introduction. *MIT Press*.

Sutton, R., McAllester, D. and Mansour, S., 1999. Policy gradient methods for reinforcement learning with function approximation. *Neural Information Processing Systems* 12, pp. 1057–1063.

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* pp. 31–36.

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

van Hasselt, H., Guez, A. and Silver, D., 2016. Deep reinforcement learning with double q-learning. *AAAI Conference on Artificial Intelligence* pp. 2094–2100.

Zhang, J., Springenberg, J., Boedecker, J. and Burgard, W., 2017a. Deep reinforcement learning with successor features for navigation across similar environment. *International Conference on Intelligent Robots and Systems* pp. 2371–2378.

Zhang, J., Tai, L., Boedecker, J., Burgard, W. and Liu, M., 2017b. Neural slam: Learning to explore with external memory. *arXiv preprint*.