REINFORCEMENT LEARNING HELPS SLAM: LEARNING TO BUILD MAPS

N. Botteghi[1,*], B. Sirmacek[2], R. Schulte[1], M. Poel[3], C. Brune[4]

[1] Robotics and Mechatronics, Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, The Netherlands, n.botteghi@utwente.nl
[2] Jönköping AI Lab (JAIL), School of Engineering, Jönköping University, Sweden, beril.sirmacek@ju.se
[3] Datamanagement and Biometrics, Faculty of Electric Engineering, Mathematics and Computer Science, University of Twente, The Netherlands, m.poel@utwente.nl
[4] Applied Mathematics, Faculty of Electric Engineering, Mathematics and Computer Science, University of Twente, The Netherlands, c.brune@utwente.nl

KEY WORDS: Simultaneous Localization and Mapping, Reinforcement Learning, Autonomous Exploration, Indoor Environments

ABSTRACT:

In this research, we investigate the use of Reinforcement Learning (RL) for an effective and robust solution for exploring unknown and indoor environments and reconstructing their maps. We benefit from a Simultaneous Localization and Mapping (SLAM) algorithm for real-time robot localization and mapping. Three different reward functions are compared and tested in different environments with growing complexity. The performances of the three different RL-based path planners are assessed not only on the training environments, but also on an a priori unseen environment to test the generalization properties of the policies. The results indicate that RL-based planners trained to maximize the coverage of the map are able to consistently explore and construct the maps of different indoor environments.

1. INTRODUCTION

Simultaneous Localization and Mapping (SLAM) is the given name for the mathematical methods aiming at simultaneously building the map of the environment and estimating the pose of the given sensor within it. Even though many SLAM solutions have been proposed for decades (Dissanayake et al., 2000), (Dissanayake et al., 2001), (Mur-Artal et al., 2015), (Cadena et al., 2016), it is still an open and challenging problem due to complexity of the mathematics which generally needed to be computed recursively, accurately and in real-time. Thus, especially in the most recent years, SLAM researchers focus on finding intelligent ways (thanks to Artificial Intelligence) to keep the mathematical processes less complex yet more accurate.
The significant improvement in the SLAM algorithms has been with the introduction of the use of the extended Kalman filter (EKF) for incrementally estimating the posterior distribution over robot pose along with the positions of the landmarks. One of the first implementation can be attribute to (Smith et al., 1990) and, only later, SLAM algorithms started to be developed for mobile robotics (Montemerlo et al., 2002). Seeing the effectiveness of these new SLAM algorithms on mobile robots, researchers started to look at ways to scale up these solutions to larger and more complex environments in which there are hundreds of landmarks and obstacles (Guivant and Nebot, 2001), (Leonard and Feder, 2000), (Lu and Milios, 1997). At this point, keeping the landmarks in the memory and rapid data association became a bigger challenge (Thrun et al., 1998). Online SLAM methods therefore increasingly became a focus on selecting the latest relevant landmarks and keep positioning a mobile robot in a large environment with simpler calculations at each SLAM iteration (Smith et al., 1990).
The above mentioned SLAM algorithms are evaluated in environments which are 'known'. In other words, the 'state transition' is calculated based on the pre-coded 'control signals' in order to reach a specific goal (i.e. 'turn left if you recognize the corner landmark', 'go straight if you recognize the corridor landmark', etc.). When the environment is 'unknown' (without any existing map), pre-determination of such control signals becomes impossible. In large and complex environments, the navigation skills of mobile robots suddenly become crucial. Researchers started to develop more intelligence on these robots for exploring the environments instead of following the pre-defined paths studied in the case of the known environments. Exploration algorithms are therefore introduced to help robots to generate the right control inputs (e.g. wheel velocities) in order to navigate unknown environments and jointly build the map of the environments. When the SLAM algorithm is used in real-time for actively planning robot paths while simultaneously building the environment map, the SLAM algorithm is named as Active SLAM (Trivun et al., 2015). The most widely used Active SLAM method so far is called frontier-based exploration (Yamauchi, 1997) which generates the control signals simply looking at whether the selected frontier (i.e. region in between the known and unknown areas of the map) candidates are leading to an occupied space or not. However, this approach also comes with limitations such as not enabling to search for an optimal path (i.e. a sequence of optimal control functions for transitioning from one state to another) and relies on greedy criteria to select the frontier to be visited (e.g. minimum distance or information gain). Furthermore, a path planner for navigating to the chosen frontier without collisions is required.
Herein, we propose an intelligent SLAM solution for autonomous exploration of unknown environments using a mobile robot and Reinforcement Learning (RL) (Sutton and Barto, 2018). We look at the control signal generation and state transition of SLAM as solving a RL problem, where the agent has to learn the best sequence of actions (the best sequence of control signals), i.e the sequence that maximizes the total cumulative reward. In this context, the goal is to explore and complete the map without collisions in the minimum time. In order to choose the best action u, the agent receives information about the state of the environment and the reward r associated to it.It is worth mentioning that the notion of state in SLAM algorithms does not necessarily coincide with the notion of state in RL. The state vector in RL can

*Corresponding author.

include sensor readings, map information, actions taken by the robot in the past, etc., i.e. all the important information for learning the optimal trajectories. Because of the high number of degrees of freedom, it is difficult to choose a proper state vector, but an informative state vector is crucial for a good performance of the learning algorithm. The same reasoning holds for the reward function. The proper choice of the reward function and the state vector is the focus of this work.
The rest of the paper is organized as follows: in Section 2. related work on Reinforcement Learning navigation and exploration is presented, then the theoretical background regarding the chosen SLAM and RL algorithms in introduced in Section 3., followed by the description of the proposed approach in Section 4.. Section 5. explains the design of the experiments, Section 6. presents results and Section 7. discusses them. Eventually, Section 8. concludes the paper.

2. RELATED WORK

2.1 Robot Navigation and Active SLAM with RL

In recent years, RL has been used to learn path planning skills for autonomous robot navigation in unknown environments. Usually, these methods don't build and use any map for navigating in the different environments, e.g. (Wu et al., 2018), (Tai et al., 2017), (Zhelo et al., 2018), (Pfeiffer et al., 2017) and (Zhang et al., 2018). While these map-less approaches obtained promising results, these methods tend to be rather sample-inefficient and a long training phase is required. On the other hand, a few methods combined RL and SLAM for learning how to navigate in unknown environments, e.g. (Zhang et al., 2016), (Brunner et al., 2017), (Mustafa et al., 2019) and (Botteghi et al., 2020), with higher performances and efficiency by exploiting the knowledge stored in the maps.
Although navigation using RL is a well-studied and tackled problem, the exploration using RL for constructing maps is still under investigated. In (Kollar and Roy, 2008), RL is used to optimize the trajectories to improve the quality of the map assuming a ground truth map is available. In (Chen et al., 2019), a RL agent is trained to build 3D maps using RGB-D sensor, however expert trajectories are required to initialize the agent's policy. Eventually, in (Dooraki et al., 2018), the proposed method learns a RL-planner for navigating without collisions and constructs maps of very simple environments.

3. THEORY

3.1 SLAM: Rao-Blackwellized particle filter

To construct the maps of the environments, a Rao-Blackwellized particle filter (RBPF) is used. RBPF estimates the robot's pose independently from the posterior of the map (Murphy, 1999). This is shown in Equation (1):

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

where $x_{1:t}$ is the robot's trajectory, $z_{1:t}$ is the sequence of sensor measurements, u_{t-1} is the control input at the previous time step and m is the map of the environment.
With RBPF, the estimation of the joint posterior can be divided into two different objectives. The first one is the pose estimate using particle filter and the second is the map estimate given known poses.

3.1.1 Particle Filter Estimation

The pose of the robot $p(x_{1:t}|z_{1:t}, u_{t-1})$ is estimated using a finite set of weighted particles (2).

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

where each particle $x_t^{(n)}$ represents a possible value of the true state time step t. The new population of particles $\mathcal{X}_t$ is constructed recursively from the previous generation $\mathcal{X}_{t-1}$ by sampling from the probabilistic odometry motion model $p(x_t^{(n)}|x_{t-1}^{(n)}, u_{t-1})$. Then, by incorporating the probabilistic observation model $p(z_t|x_t^{(n)})$, an individual importance weighting factor $w_t^{(n)}$ is assigned to each particle. Eventually, resampling takes place and the particles with the smaller importance weights are not resampled allowing the filter to converge to the correct pose estimate (Grisetti et al., 2005).

3.1.2 Mapping with Known Poses

After estimating the pose of the robot, the posterior of the map $p(m|x_{1:t}, z_{1:t})$ is estimated. This step is called "mapping with known poses" (Moravec, 1988). The map is assumed to be an occupancy grid: the environment is split into evenly spaced cells and a probability value is assigned to each cell to indicate if it is occupied, free or unknown. The probability of each grid cell is chosen independent from the others. By doing that, the map's posterior can be computed as the product of the single cells posterior of the map m_i as shown in Equation (3),

$$p(m|z_{1:t}, x_{1:t}) = \prod_{i=0}^{M} p(m_i|z_{1:t}, x_{1:t}) \ , \tag{3}$$

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

3.2 Reinforcement Learning

Reinforcement Learning (RL) (Sutton and Barto, 2018) aims at solving a sequential decision making problem by learning the actions to take through the interaction with an unknown environment. The RL-agent, i.e. the decision maker, is not taught what is the best behaviour, but it has to infer it by only receiving a scalar value, i.e. the reward, for each action taken. The interaction between the agent and environment can be modelled as a Markov Decision Process (MDP) (Sutton and Barto, 2018). A MDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where $\mathcal{S}$ is the set of possible states, $\mathcal{A}$ is the set of actions, $\mathcal{P}$ describes the dynamics of the environment, i.e. the transition probability from a state s_t to the next next state s_{t+1} when applying an action a_t and a reward function $\mathcal{R}$. Usually $\mathcal{P}$ and $\mathcal{R}$ are unknown beforehand and has to be learned by trial and error.
Active-SLAM aims at finding the sequence of control inputs, i.e. actions, to explore and fully construct the maps of the environments. This sequential decision making process can be formulated as a RL problem, where the RL-agent has to learn the actions that maximize the cumulative reward in Equation 4.

$$R(s) = \Sigma_{t=0}^{T} r_{t+1} \tag{4}$$

where $R(s)$ is the reward function that we assume is only dependent on the state. In this case the maximum of the reward function corresponds to the full completion of the maps, that we assume represented as a 2D occupancy grid. RL is suitable for this application as no ground truth information is required beforehand (as it happens with supervised learning) and the exploration policy can be naturally learned by the trial and error interaction between the agent and the (unknown) environment. Thanks to that, RL is able to solve decision processes without any knowledge of the state transition function.

3.2.1 Deep Q-Network For choosing the best action, the so-called value function is estimated and employed by many RL-algorithms. The value function contains information on the quality and desirability of the states. Deep Q-Network (DQN) (Mnih et al., 2013) estimates the state-action value function Q, i.e. a function measuring how good it is to choose a certain action in a given state, by means of a neural network. It is common to employ function approximators when the state space and/or the action space are continuous and cannot be represented by a look-up table. In particular, the Q-network is trained to minimize the mean square error between the true state-action value function and its prediction, as shown in Equation (5).

$$L\left(\theta^{Q}\right) = \mathbb{E}\left[\left(\underbrace{Q\left(s_t, a_t|\theta^{Q}\right) - Q_{target}}_{\text{TD error}}\right)^2\right] \tag{5}$$

where $Q_{target} = r(s_t, a_t) + \gamma \max_a Q\left(s_{t+1}, a_t|\theta^{Q}\right)$.

Despite the advantages of handling continuous state spaces, the neural network, representing the value function, introduces training instabilities as the samples are not identically and independently distributed (i.i.d. assumption), but collected from correlated trajectories (e.g. consider the case of mobile robot navigation). This problem is solved with the introduction of the Experience Replay (Lin, 1993): the training batches are constructed of randomly selected samples to break any temporal correlation. Furthermore, the value $\max_a Q(s_{t+1}, a|\theta^{Q})$ used to compute the Q_{target}, and consequently used to train the Q-network, shouldn't be generated by the same neural network that is trained. This cause an overestimation of the Q-value of the next state. This problem is addressed and solved by Double DQN (DDQN)
(Van Hasselt et al., 2016), where a target network, parametrized by θ^{t}, is used to compute the $\max_a Q(s_{t+1}, a|\theta^{t,Q})$.

3.2.2 Deep Recurrent Q-Network DQN has proven to be successful in many applications, however, most of these rely on the Markov assumption that the future is independent on the past given the present, i.e. the information carried by the state at the current time step is enough for choosing the best action. In many situations, the state of the environment is not directly observable, i.e. in the case of Partially Observable MDPs (e.g. two sensor readings may look the same, but they correspond to two different situations), and DQN struggles in learning the optimal policy. To tackle this problem Deep Recurrent Q-network (DRQN) (Hausknecht and Stone, 2015) was introduced. DRQN extends DQN to partially observable environments by adding a recurrent neural network (RNN) in the Q-network.

4. METHODOLOGY

We phrase the mapping of indoor and unknown environments as a finite horizon problem, by assuming that a terminal goal-state exists for the early termination of the episodes and it corresponds to the completion of the map.
We aim at strengthening the combination of RL and SLAM, in order to build maps in the most efficient way. To do that we focus on two crucial aspects of RL: the reward function and the state space.

4.1 Reward shaping

The choice of the reward function is an important aspect in RL, as it should incorporate task-specific knowledge that the agent uses to learn the optimal behaviour. Here we compare three different reward functions:

1. Sparse (in Equation (6))
2. Map-completeness (in Equation (7))
3. Information-gain (in Equation (9))

The first straightforward option for learning to map an unknown environment is a "sparse" reward function, in Equation (6). By learning to avoid collision, the agent may occasionally complete the map. This will be used as a baseline for comparison with the other reward functions.

$$R(s_t) = \begin{cases} r_{\text{mapCompleted}}, & c_t \geq \bar{c}, \\ r_{\text{crashed}}, & s_{coll}, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

where $r_{\text{mapCompleted}}$ is a positive scalar when the map, indicated by c_t [1], depending on a scaling factor λ, is completed about a certain threshold $\bar{c}$ and r_{crashed} is a negative scalar if a terminal collision-state s_{coll} is reached, i.e. if a collision occurs.
The second reward exploits and includes more information about the map, in particular it adds to the "sparse" reward a term proportional to the difference in the map completeness of the current time step and the previous time step.

$$R(s_t) = \begin{cases} r_{\text{mapCompleted}}, & c_t \geq \bar{c}, \\ r_{\text{crashed}}, & s_{coll}, \\ c_t - c_{t-1}, & \text{otherwise.} \end{cases} \tag{7}$$

where c_t is the map completeness at time t and c_{t-1} is the map completeness at time $t-1$.
The third reward term utilizes the map's entropy (see Equation (8)). The map's entropy (Thrun et al., 1998) corresponds to the sum of the entropy of all the cells c in the map m and it quantifies the uncertainties in the map. A good exploration policy is a policy that explores the environment and reduces the uncertainties.

$$H(m) = \sum_{c_f \in m} p(c)\log p(c) + \sum_{c_o \in m} (1-p(c_o))\log(1-p(c_o)) \tag{8}$$

where c_f corresponds to the unknown and unoccupied cells in the map and c_o to the occupied ones. Analogously to map-completeness reward, the complete reward function adds to the "sparse" reward function the difference in entropy at the current time step and at the previous one, i.e. the information-gain.

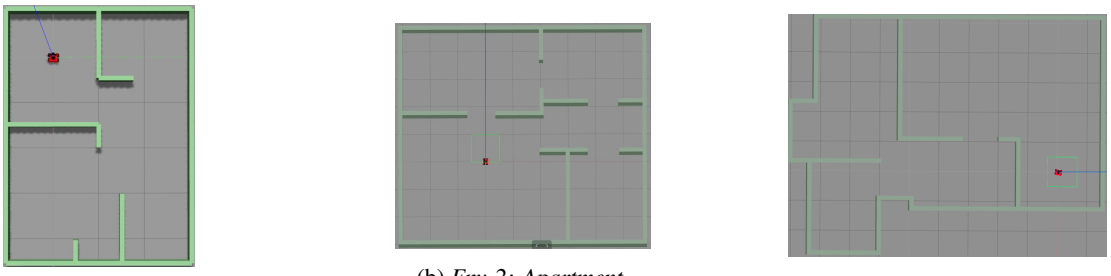
$$R(s_t) = \begin{cases} r_{\text{reached}}, & c_t \geq \bar{c}, \\ r_{\text{crashed}}, & s_{coll}, \\ H_t - H_{t-1}, & \text{otherwise.} \end{cases} \tag{9}$$

where H_t is the map's entropy at time t and H_{t-1} is the map's entropy at time $t-1$.

4.2 State and Action spaces

The agent chooses its actions based on the information contained in the state vector. A state vector must contain all the relevant information for solving a given task. In particular, we analysed two different compositions of state vectors, namely one containing only the present information at time t (see Equation 10) and

[1] The map completeness is naturally defined as $c_t = \lambda \frac{U+O}{T}$ where U is the number of unoccupied cells in the map, O is the number of occupied cells, T the total number of cells and λ is a scaling factor, such that $c_t \in [0, 1]$, based on the actual area of the environment to explore



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

Figure 1: The environments are of size 22, 68 and 65.5 m^2, respectively.

one containing also information of the recent past (see Equation 11):

$$s_t = [z_t, a_{t-1}, \hat{x}_t, c_t, t_t] \tag{10}$$

where s_t corresponds to the state vector at time t, z_t to the current sensor readings, a_{t-1} to the previously chosen action, $\hat{x}_t$ to the robot's pose estimate, c_t to the completion percentage of the map and t_t to the time steps left before the end of the episode.

$$s_t^e = [s_{t-w}, ..., s_t] \tag{11}$$

where s_t^e is the extended state vector that concatenates a sequence of past states s from Equation (10) and w is the length of the observation window i.e. how much of the past is taken into account. The agent is not allowed to observe the full map because grid-maps are variable dimension vectors that depend on the overall size of the environment which is unknown beforehand. We would need to allocate enough space in the state vector for any kind of map independently of their actual sizes. This would make the state vector explode in dimension and it would require the use of deeper neural networks for representing the policies. These conditions would make the learning of the optimal behaviour harder. For sake of simplicity, we employ a discrete action space. The agent can choose between four possible actions: go forward, go backward, turn left and turn right. However, the proposed approach can be extended to continuous action spaces.

5. EXPERIMENTS DESIGN

The experiments are performed in a 3D simulation environment, using the Robot Operating System (ROS) middleware and the Gazebo simulator, with a differential drive mobile robot (Husarion), controlled through velocity commands and equipped with 360° laser range scanner (LiDAR) and wheels odometry.

5.1 Comparison of the different reward functions

We aim at evaluating the three proposed reward functions (Section 4.1) in three different environments, shown in Figure 1. In particular, the environments, depicted in Figure 1b and 1c, were created on the base of actual apartments in the Netherlands (funda.nl, 2020). We train our agents in *Env-1* and *Env-2* and we compare the learning performances of the different reward functions. Then, the trained agents are evaluated based on the quality of the trajectories generated and further assessed in the unseen a priori environment *Env-3*. Furthermore, we compare our method with frontier-based exploration [2] (Yamauchi, 1997).

[2]We use ROS package *explore-lite* for the implementation of the frontier-based exploration.

5.1.1 Different handling of the robot collisions

We propose two different ways to handle the collisions of the robot during the training and testing phases.
First, when a collision occurs, i.e. when the distance between the robot and an obstacle is smaller than a given threshold (see Table 2 in Appendix), the episode terminates and the $r_{crashed}$ penalty is applied. Alternatively, when in a collision state, no reset of the episode occurs, but the $r_{crashed}$ penalty is still applied. In this case, each episode lasts the same amount of time-steps.

5.2 Use of the memory in neural network architecture

We compare two different algorithms: DQN and DRQN. The main difference lies in the use of the memory in the neural network representing the state-action value function Q. For DQN, we employ a simple neural network with 2 fully connected layers with 512 neurons each and a fully connected output layer that outputs a Q-value per action. DRQN, instead, utilizes a Long Term Short Memory (LSTM) layer, a fully connected layer with 512 neurons and again a fully connected layer outputting the Q-values.

6. RESULTS

6.1 Comparison of the different reward function

We compared different reward functions (see Equation (6)-(9)) in *Env-1* and *Env-2* with the two different methods of handling collisions (Section 5.1.1). In Figure 2, the results are presented. In particular, Figure 2a and 2b show the training performances in *Env-1* with respect to the map completeness when no-resetting on wall contact and when resetting. In both cases, when a collision occurs a penalty reward is received by the agents. By avoiding the reset when an obstacle is hit, the agent can better explore the environment while learning to avoid collisions. When resetting, the agent has to first learn to perfectly avoid collisions before being able to escape the starting room and collect good samples. In Figure 2c and 2d, the performances of the agents trained with the different reward functions in *Env-2* are shown. For mapping an unknown environment, early stopping is not beneficial and it increases the probability of getting stuck in local optimal policies (e.g. circling in one room). Furthermore, the reset generates undesirable higher variance of the map completeness signal over training. As *Env-1* is rather simple and the main challenge is escaping the main room, there is not a significant difference between the three proposed reward functions (see Figure 2a). However, as soon as the environment gets more complex, the agent trained with the "sparse" reward function (6) struggles to complete the map (2c). Moreover, the agent trained with the "map-completeness" reward (7) seems to converge faster than the agent trained with the "information-gain" one (9).

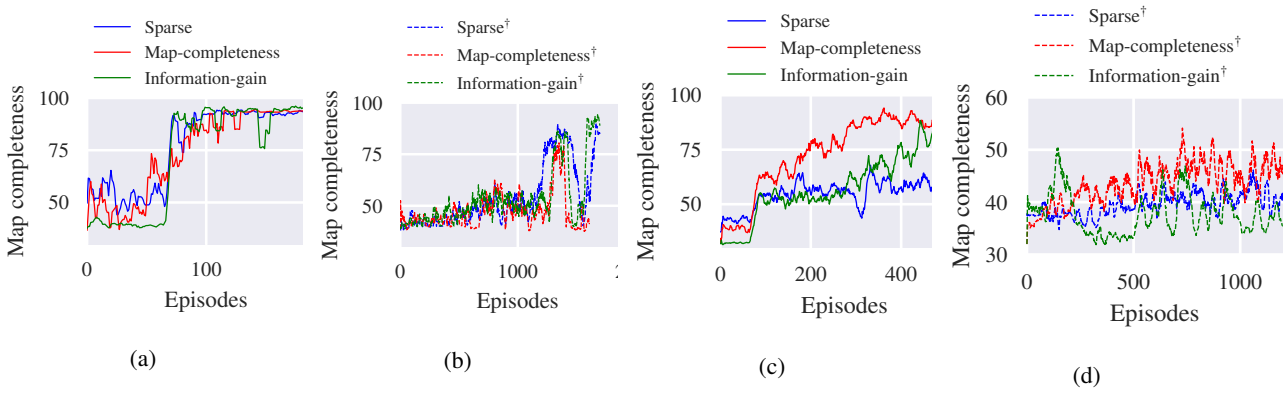


Figure 2: Comparison of the different reward functions on Env-1, when no-resetting on collisions (Figure 2a) and when resetting (Figure 2b) and comparison of the different reward functions on Env-2, when no-resetting on collisions (Figure 2c) and when resetting (Figure 2d).

6.2 Use of the memory in neural network architecture

We study and analyse the effect of the memory in the state vector and to what extend it can be useful for improving the performances of the RL-agent. In particular, we compare on *Env-1* (Figure 1a) the agent using no memory, i.e. DQN, and the agent using memory i.e. DRQN, when the reward function in Equation (9) is used. The progresses of the map completeness over the training episodes for the two agents is shown in Figure 3.
DQN starts learning sooner than DRQN as it requires the collection of a number of samples equal to the training-batch size (see Table 2), while DRQN needs to wait until the same number of sequences of length equal to the window size (see Table 2) are collected. When the actual training of the neural network starts, DRQN exhibits a steeper learning curve than DQN and a smaller variance of the map completeness during training. Nevertheless, both agents converge to the same map completeness.

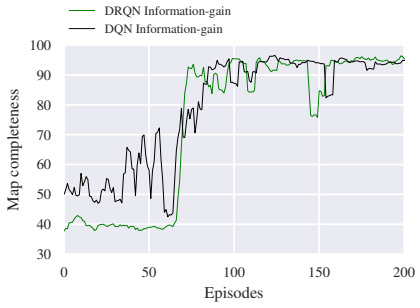


Figure 3: Comparison of training performance of DQN and DRQN when the "information-gain" reward function (9) is employed.

Completing the map involves delayed reward as several actions may lead to no variation in the reward (e.g. circling around in the same room), thus using a sequence of observation is beneficial for solving the mapping task.

6.3 Evaluation of the trajectories

In this section, the trajectories followed by the agents when no noise is applied to the actions are shown and analysed. In particular, the trajectories generated by the policies trained with the three proposed reward functions are compared to the ones generated using the frontier-based exploration methods (Yamauchi, 1997). The trajectories are depicted in Figure 4. In the training environments (see Figure 4a and 4b), the RL-planners trained with "map-completeness" (7) and "information-gain" (9) reward functions complete the maps by following shorter trajectories than the ones generated by the frontier-based exploration algorithm. The "sparse" reward function (6), that uses none of the knowledge stored in the map, fails to complete the map as soon as the environment grows in complexity. In an a priori unseen environment (see Figure 4c and 4d), only the "map-completeness" (7) and "information-gain" (9) can complete the map, but the performances are influences and dependent on the starting position of the robot. In particular, the agent trained with "information-gain" reward performs better when initialized in a clustered space as it naturally tends to navigate to open areas. On the other hand, the agent trained with "map-completeness" reward exhibits the opposite behavior, as it tends to stay close to walls and obstacles. The paths length during the evaluation experiments was recorded and it is shown in Table 1.

Approach	Env-1	Env-2	Env-3[1]	Env-3[2]
Sparse	5.0 m	40.0 m*	44.9 m*	32.7 m*
Map-completeness	4.7 m	11.6 m	34.7 m*	35.1 m
Information-gain	4.1 m	11.9 m	28.5 m	109.9 m*
Frontier	3.9 m	20.2 m	14.8 m	15.5 m

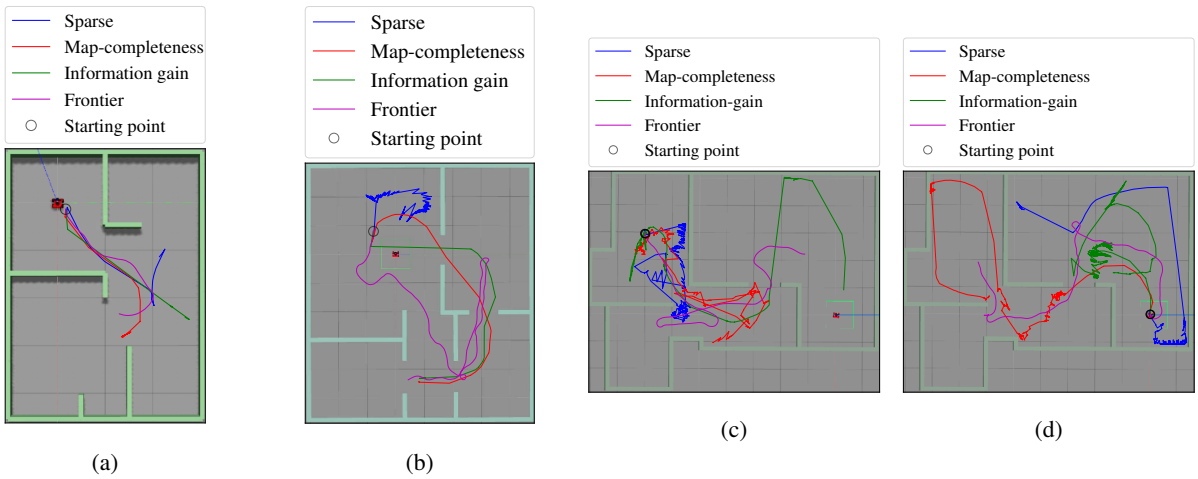
* the map is not completed.
[1] starting position of the robot in the top-left room.
[2] starting position of the robot in the bottom-right room.

Table 1: Trajectory length comparison in the different environments.

7. DISCUSSIONS

7.1 Mapping with RL

RL allows to learn smooth trajectories for optimizing the completion of maps in environments with different topologies. When the reward function exploits the information stored in the map, the agent is able to learn short paths to complete the map. Reward functions based on map-completeness and information-gain outperform the sparse reward function, especially when the environment is more complex (e.g. *Env-2* and *Env-3*). In particular, the agent that is trained to maximize information-gain learns and follows safer and longer paths and tends to navigate towards open areas (e.g. center of the rooms) than the agent trained to maximize the map completeness.



(a) (b) (c) (d)

Figure 4: Comparison of trajectories generated by the RL-planners and the frontier-based exploration algorithm in the three environments.

No initial information about the environment, nor a map is required in the state vector. Thus, we are able to use a simple neural network structure with fewer parameters, which trains faster and requires less samples than a deeper architecture. Adding a memory component (i.e. LSTM) is the neural network architecture is beneficial for improving the performances of the agents. However, the MDP modelling framework is violated and a POMDP framework should be used instead.
The paths generated by the RL-agents are shorter than the ones obtained using frontier-based exploration method in the training environments. In unseen a priori environment, RL-methods often complete the maps, but they seem influenced by the starting position of the robot.

7.2 Future work

First, we aim at testing the proposed approach in bigger and more complicated environments (e.g. the environments proposed in (Khoshelham and Acharya, 2017)).
Secondly, we phrased the mapping problem as a finite horizon one by assuming the knowledge of the overall area of the environment. This can be practical in many application, but it would be interesting to relax this hypothesis and utilize an infinite horizon framework (i.e. no knowledge on the total area of the environment is needed). This would require the learning of higher-level exploration skills as no early termination is present.
Here we didn't investigate whether our RL-SLAM solution can solve the loop closure problem, however this is another interesting direction for future work.
Furthermore, we aim at transferring the policies learned in the virtual and simulated environment to real robots, similarly to the approach followed in (Mustafa et al., 2019).
Eventually, we would like to extend our RL-SLAM solution to multi-agent exploration by looking at optimal solutions for rapid exploration of unknown environments.

8. CONCLUSIONS

Herein we proposed a new approach combining RL and SLAM for building maps of indoor environments in an efficient way.
In the training environments, the RL-agents that exploit the information stored in the map, i.e. "map-completeness" (7) and "information-gain" (9), outperform the "sparse" (6) reward function and the frontier-based exploration approach in term of path length and consequently time to complete the map. The map-completeness reward function (7) achieves the fastest learning curve over training, while the information-gain based reward (9) allows the agent to learn the safest exploratory path. In an a priori unseen environment with similar total area, but different topology, the RL-planners trained with (7) and (9) are able to complete the map. However, both planners follow longer paths than frontier-based exploration.
Furthermore, the use of the memory in the RL-algorithm is beneficial for the performances of the agents.
The proposed RL-framework is suitable for generating exploratory trajectories to efficiently construct maps using SLAM algorithms. Transferring the policies learned in the virtual environments to real robots is the next step for the deployment of this framework in real life scenarios.

REFERENCES

Botteghi, N., Sirmacek, B., Mustafa, K. A. A., Poel, M. and Stramigioli, S., 2020. On reward shaping for mobile robot navigation: A reinforcement learning and slam based approach.

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

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I. and Leonard, J., 2016. Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age. IEEE Transactions on Robotics 32(6), pp. 1309–1332.

Chen, T., Gupta, S. and Gupta, A., 2019. Learning exploration policies for navigation. ICLR.

Dissanayake, G., Newman, P., Clark, S., Durrant-Whyte, H. F. and Csorba, M., 2001. A solution to the simultaneous localization and map building (slam) problem. IEEE Trans. Robotics and Automation 17, pp. 229–241.

Dissanayake, M. W. M. G., Newman, P., Durrant-Whyte, H. F., Clark, S. and Csorba, M., 2000. An experimental and theoretical investigation into simultaneous localisation and map building. In: Experimental Robotics VI, Springer London, London, pp. 265–274.

Dooraki, R., Amir, Lee and Deok-Jin, 2018. An end-to-end deep reinforcement learning-based intelligent agent capable of autonomous exploration in unknown environments. Sensors 18(10), pp. 3575.

funda.nl, 2020. https://www.funda.nl/. Houses in the Netherlands, [Online; accessed April 2020].

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.

Guivant, J. and Nebot, E., 2001. Optimization of the simultaneous localization and map-building algorithm for real-time implementation. IEEE transactions on Robotics and Automation 17, pp. 242–257.

Hausknecht, M. and Stone, P., 2015. Deep recurrent q-learning for partially observable mdps.

Khoshelham, K., D. V. L. P. M. K. Z. and Acharya, D., 2017. Isprs benchmark on indoor modelling. Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci. XLII-2/W7, pp. 367–372.

Kollar, T. and Roy, N., 2008. Trajectory optimization using reinforcement learning for map exploration. The International Journal of Robotics Research 27(2), pp. 175–196.

Leonard, J. J. and Feder, H. J. S., 2000. A computationally efficient method for large-scale concurrent mapping and localization.

Lin, L.-J., 1993. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science.

Lu, F. and Milios, E. E., 1997. Globally consistent range scan alignment for environment mapping. Autonomous Robots 4, pp. 333–349.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

Montemerlo, M., Thrun, S., Koller, D. and Wegbreit, B., 2002. Fastslam: A factored solution to the simultaneous localization and mapping problem.

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

Mur-Artal, R., Montiel, J. and Tardos, J., 2015. Orb-slam: a versatile and accurate monocular slam system. IEEE Transactions on Robotics 31, pp. 1147 – 1163.

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

Mustafa, K. A. A., 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, pp. 857–863.

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).

Smith, R., Self, M. and Cheeseman, P., 1990. Estimating Uncertain Spatial Relationships in Robotics. Springer New York, New York, NY, pp. 167–193.

Sutton, R. S. and Barto, A. G., 2018. Reinforcement Learning: An Introduction. Cambridge, MA : The MIT Press.

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., 1998. A probabilistic approach to concurrent mapping and localization for mobile robots. Autonomous Robots 5, pp. 253–271.

Trivun, D., Salaka, E., Osmankovic, D., Velagic, J. and Osmic, N., 2015. Active slam-based algorithm for autonomous exploration with mobile robot. 2015 IEEE International Conference on Industrial Technology (ICIT) pp. 74–79.

Van Hasselt, H., Guez, A. and Silver, D., 2016. Deep reinforcement learning with double q-learning.

Wu, Y., Wu, Y., Gkioxari, G. and Tian, Y., 2018. Building generalizable agents with a realistic and rich 3d environment.

Yamauchi, B., 1997. A frontier-based approach for autonomous exploration. Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation' pp. 146–151.

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

Zhang, W., Zhang, Y. and Liu, N., 2018. Danger-aware adaptive composition of drl agents for self-navigation.

Zhelo, O., Zhang, J., Tai, L., Liu, M. and Burgard, W., 2018. Curiosity-driven exploration for mapless navigation with deep reinforcement learning.

APPENDIX

The parameters used in the experiments are shown in Table 2.

RL and SLAM parameters	Value
training steps	500000
optimizer	ADAM
DQN learning rate	10^{-3}
DRQN learning rate	10^{-4}
L2-regularization coefficient	10^{-2}
discount factor γ	0.99
ϵ-greedy coefficient	0.6
ϵ-decay	0.9997
min. ϵ-greedy coefficient	0.1
DQN batch size	64
DRQN batch size	64
LSTM window size	5
particles	50
process scan threshold translation	0.05
process scan threshold rotation	0.05
grid cell size	$0.05\,\mathrm{m} \times 0.05\,\mathrm{m}$
occupancy threshold	0.65
LiDAR max. range	$4\,\mathrm{m}$
LiDAR min. range	$0.2\,\mathrm{m}$
collision threshold	$0.2\,\mathrm{m}$

Table 2: Parameters of the experiments.