Applying PAIRED to Automated Map Generation

Lisa Golla
Cognitive Science
Universität Osnabrück
Osnabrück, Germany
lgolla@uni-osnabrueck.de

Hannah Köster
Cognitive Science
Universität Osnabrück
Osnabrück, Germany
hakoester@uni-osnabrueck.de

***Abstract*—Robotic mapping remains a challenge in the field of robot navigation. Whereas non-DRL approaches suffer from high computational complexity and from the need of certain assumptions, DRL-based end-to-end approaches can provide a solution. We faced the task of automatic map generation by employing the PAIRED algorithm, which was only used for solving grid worlds and web navigation so far. On behalf of focusing on the performance, we decided to choose grid worlds as an environmental setup and have excluded the task of localizing. We exposed that the algorithm can be used for map generation and that it performs quite well. Applying two optimizations, flexible and budget PAIRED, to the vanilla PAIRED separately as well as in combination revealed that the combination, flexible b-PAIRED, showed the best performance. For further experiments the source code is available for reproducibility.**

***Index Terms*—autonomous robotic mapping, PAIRED algorithm, DRL, SPLAM**

I. Introduction

Robot navigation is a current topic. For example, robot vacuum cleaners need to navigate through a given environment and to reconstruct a map accordingly. In certain restaurants robots are already used to transport the cooked food to the respective table, where the robot has to manage the navigation. The industry uses robots in the field of logistics when certain objects have to be moved or placed differently. If for example humans are part of the environment one cannot apply closed-loop systems anymore which demonstrates the need for RL solutions. Again, this requires the ability of the robot to interact with dynamic environments accordingly. There are various applications where robot navigation is in use, which highlights the importance of the given research field.

One of the major challenges when it comes to robot navigation is robotic mapping. Having good, accurate maps is especially important, because the maps are often the basis for planning the robot's movements and also play an important part for localization. With the gold standard requiring a trained human operator to drive a robot through the environment that needs to be mapped, autonomous robotic mapping has become of more and more interest as the need for generating adequate maps for the increasing number of robots used rises. With one of the goals being for such an autonomous robot to construct a map, as well as localize itself, the discipline is linked to the field of computer vision and cartography. Recently, several non deep reinforcement learning approaches have been applied to robotic mapping [9]–[11]. These methods based on geometric reconstruction have proven to be effective for the field of navigation and collision avoidance, however they go along with considerable computational overhead, as shown in [12]. Additionally, what also has to be considered are certain assumptions, like static environments, which are necessary for these methods to work, however, these assumptions may not always hold in practice [13]. Reinforcement learning (RL) approaches can help dissolve the assumptions needed for the geometric reconstruction methods, as well as serve two main advantages [5]. Firstly, Reinforcement learning based methods may perform more intelligent actions under the condition of partial observation due to the knowledge exploitation of statistics. Secondly, as the essence of RL suggests, these methods are learning from mistakes and can therefore continuously improve their proficiency which corresponds to the well known trial and error procedure. Next to that, non-RL approaches, e.g. approaches based on information gain, may also have the disadvantage of using a two-step-approach, where two systems are used, one that chooses the next optimal sensing location and one that plans the trajectory to that location. For example, even if the agent decided on one direction, it is also necessary to let a different system calculating the navigation of the agent into that direction. In contrast, when using RL-approaches we usually use one approach to solve several issues, which is also called an end-to-end solution. Here, one system can be used to solve the next action and to plan the trajectory which may be in general a more complex solution, but has the possibility to lead to a better solution as every time step can be considered to optimise exploration [16]. In conclusion, it is worth employing and testing RL approaches in automatic map generation. Nevertheless, it is also important to keep in mind that these approaches are raising new issues which need to be solved as outlined and addressed in section II-A.

In our experiment we decided to focus on robotic mapping and to evaluate the respective, generated map. The Map is generated in a metric framework which means that it considers a two-dimensional space in which it places objects. By contrast, a so-called topological framework is only based on places and relations between them. Usually, the distances between places are stored. The map can then be considered as a graph, in which nodes can be seen as places and arcs as paths. Further, the map learning is dependent on the localization process, and a difficulty arises when errors in localization are incorporated into the map. This problem is commonly referred

to as Simultaneous Localization and Mapping (SLAM) or - if planning the path to the next location is included - Simultaneous Planning, Localization and Mapping (SPLAM). In order to overcome the SLAM problem, we decided to simplify the localization task. In our experiment, we have knowledge about how the agent is moving or rather how he is intending to move and assume that this movement to a new position is working out, without the need for the agent to check that, except for being able to detect driving against walls and assuming no movement in those cases. We also decided to employ a grid environment since learning a policy for large environments from scratch is often difficult and takes a lot of time and computational power. A more scalable approach is to learn a policy in small environments and transfer it to large environments by repeating the reasoning process. As another paper reveals, learned policies in a grid can be successfully generalized to new environments and transferred to larger, and more complex environments, as well [6]. Consequently, our learning policy has a meaning also for further research which is employing more complex, larger environments. In addition, using a grid world to test the given code has a certain advantage. Namely, small-scale environments are less computationally intensive and lend themselves well to a more critical and thorough analysis of the performance, behaviors, and intricacies of new algorithms [7], [8].

Overall, our task can be summarized as the following. We found a relevant paper which has implemented an unsupervised gridworld generation by introducing the Protagonist Antagonist Induced Regret Environment Design (PAIRED) algorithm for automatic environment generation [1]. We used the implementation of the paper and adapted it to our needs, e.g. there is no goal state in the environment that the agent has to reach, rather his task is to find an effective and robust solution for exploring the unknown and creating a map of the environment in the process. Therefore, we had to adapt the reward, goal statement and visualization part as well as the metrics that are logged in the code section. How we adapted goal and reward in our experiment will be explained in more detail in section IV. We applied two optimized variants of the PAIRED algorithm, flexible PAIRED and budget PAIRED [2], as well as the combination so-called flexible b-PAIRED. Moreover, we also tested how the vanilla PAIRED algorithm performs with and without continuously negative reward.

This paper makes the following contributions:

- Testing how the PAIRED algorithm performs regarding the task of robotic mapping
- Comparing the performance of different optimized versions of PAIRED
- Testing the influence of continuous negative reward on the performance
- Gives rise to a learned policy which can be transferred to more complex and large environments
- Introduces useful metrics to evaluate the performance of the algorithm

II. Background

A. Planning under uncertainty

Robot navigation is linked to one fundamental issue: decision making under uncertainty. Especially partial observability makes the computation harder [14], because the agent is not able to determine the exact state on the basis of observation as well as planning for optimal actions needs an integration of information over the past history of actions and observations. Concerning a model-based approach the problem could be formulated as a Partially Observable Markov Decision Process (POMDP). Whereas it is computationally intractable to solve this exactly, there are algorithms approximating POMDP which show a vast progress [6], but learning from manually constructed POMDP models still remains difficult. The model free approach searches directly for an optimal solution within a policy class. If we do not restrict the policy class, the difficulty is data and computational efficiency.

A POMDP can formally be defined as a tuple (A, O, S, T, I, R, y). A denotes the set of actions, O the set of observations and S the set of states. τ is corresponding to the transition function $\tau : S \times A \rightarrow \Delta(S)$, I is an observation function $I : S \rightarrow O$ and R $R : S \rightarrow \mathbb{R}$, whereas y is a discount factor. The utility can be defined as $U(\pi) = \sum_{i=0}^{T} r_t \gamma^t$, where T is a horizon.

B. Related Work

In order to solve the issue of learning policies for decision making in partially observable domains, [6] proposed the approach of combining model-based and model-free learning by embedding a model and a planning algorithm in a recurrent neural network (RNN) that represents a policy. The neural network architecture is called QMDP-net. A QMDP-net is a recurrent policy network, but it represents a policy by connecting a POMDP model with an algorithm, called QMDP , that solves the model, thus embedding the solution structure of planning in a network learning architecture. The network contains two modules, one is representing a Bayesian filter including the history of an agent's actions, as well as observations which are transformed into a certain probabilistic estimate of the agent's state. The other module is the QMDP algorithm which chooses a certain action on the basis of the current belief. The aim was to learn a policy that enables an agent to act effectively in a diverse set of partially observable stochastic environments. It could be shown that the network is successfully learning policies that generalize to new environments. Moreover, experiments on several simulated robotic tasks, including navigating through a grid world, show that learned QMDP-net policies successfully generalize to new environments and transfer to larger environments as well.

In comparison, [15] introduced a Deep Recurrent Q-network (DRQN) as a solution. DRQN extends DQN to partially observable environments by adding a recurrent neural network (RNN) in the Q-network since DQN struggles with learning an optimal policy when it comes to Partially Observable MDPs. As well as the QMDP-net approach, the DRQN approach,

which uses memory in the RL-algorithm, has proven to be beneficial for the performances of the agents. The paper also introduces a detailed analysis of different reward functions, whose results will be further discussed in section IV-B where we outline how this knowledge contributed to our reward definition.

The two named papers were employing RNN approaches. We based our work on the so-called PAIRED - algorithm which uses a LSTM, e.g. a RNN, approach in their further implementation, whereas theoretically any DRL algorithm could be added to the PAIRED algorithm. This may evoke the idea of testing different algorithms like QMDP and DRQN in combination with PAIRED. As we can see in the next section, the issue of learning a policy regarding the POMDPs is solved differently. Especially, what is not addressed in the other approaches is the common failure of domain randomization which cannot adapt to the agent's learning progress and the minmax adversarial training which leads to worst-case environments which are often unsolvable. We used the PAIRED algorithm to demonstrate the performance in the field of robotic mapping. As introduced in [1] and [2] the algorithm was only used for an agent to navigate from a start to a goal state in a grid world and in the context of web navigation. We postulate that the Adversarial Environment Generation is an important factor in order to make successful learning of the agent's policy possible.

III. PAIRED

Designing an appropriate distribution of environments is challenging. Real world applications are highly complex, listing every edge case that is relevant becomes impractical. The idea is to automate this process, excluding the need of providing a specified distribution. [1] proposes an approach that makes it possible to specify the domain of environments relevant for training the policy with only the need to supply an underspecified environment, modeled by an Underspecified Partially Observable Markov Decision Process (UPOMDP). This is an environment that has free parameters controlling its behavior. One example for these parameters could be the positions for blocks in an environment. The method will then construct distributions of environments by providing a distribution of settings of these free parameters. The problem of taking the underspecified environment and a policy, and producing an interesting distribution of fully specified environments in which that policy can be further trained is called Unsupervised Environment Design (UED). The aim of UED is to generate a distribution of environments which best supports the continued learning of a particular agent policy, which can be described as the environment policy. PAIRED is a method approximating the minimax regret environment policy.

The proposed algorithm is called Protagonist Antagonist Induced Regret Environment Design (PAIRED). With regret the difference between the payoff obtained for a decision, and the optimal payoff that could have been obtained in the same setting with a different decision is meant. For the purpose of approximating this regret 2 agents are trained under the same environment conditions and the difference of their payoffs are used:

$$REGRET^{\overrightarrow{\theta}}(\pi^P, \pi^A) = U^{\overrightarrow{\theta}}(\pi^A) - U^{\overrightarrow{\theta}}(\pi^P)$$

An environment adversary is introduced which is learning to control the parameters of the environment in a way that the regret of the protagonist is maximized. By contrast, the protagonist tries to minimize the regret and the antagonist also aims to maximize the regret. Beneficial of this approach is that since the adversary is motivated to challenge the protagonist in a way that it creates the easiest task on which the protagonist fails and the antagonist succeeds, feasible, but not unsolvable environments are generated. A too complex environment would yield a negative reward, since both agents would perform in a bad way.

Since PAIRED only demonstrated results on simple grid-world environments [1], and did not expand to complex, high-dimensional state-action space, a more flexible estimate of the regret, as well as a budget mechanism [2] was introduced to improve the PAIRED algorithm.

A. Flexible PAIRED

The PAIRED algorithm succeeds, if the adversary and antagonist coordinate and reach a Nash equilibrium with the protagonist, then the protagonist will have learned to minimize the regret. However, as revealed in [2], in practice it often fails to converge. In the case where the algorithm fails, PAIRED minimizes regret with respect to the antagonist's policy, which means the protagonist will learn to be as good as the antagonist. As a matter of fact, the adversary cannot continue to train the protagonist and the antagonist fails to improve. Flexible PAIRED approaches this issue. Here, the objective does not make a distinction between antagonist and protagonist agents, and instead annotates the best performing agent as the antagonist.

$$REGRET = max\{R^A, R^P\} - 0.5 * (R^A + R^P)$$

On the condition that the one agent has a higher performance than the other agent, the objective will continuously improve the weakest agent. Meanwhile, the other agent in the policy continues learning, and therefore provides a stronger maximum performance against which we measure the regret.

B. Budget PAIRED

Another issue of PAIRED which is addressed in the Budget PAIRED approach is that small, non-informative regret will hinder the adversary's ability to design environments at an appropriate difficulty for agents to learn. On this account, a budget is introduced which is enforcing the objective that binds the adversary's design budget to the performance of the best agent in addition to the regret. In [2], where budget PAIRED is introduced, the adversary has to built increasingly complex websites for the agents to navigate by adding design primitives. As a set number of primitives has to be added, there

is also the option to add a SKIP action, which does nothing. The budget can be approximated as the expected number of non-SKIP actions over N time steps. Here, this budget is updated accordingly if the agents are learning. Consequently, the adversary is encouraged to use more budget (less SKIP actions) when the navigator agents are accumulating more positive rewards in the environment and respectively, less budget is used when negative reward is collected.

$$O_{budget} = R^A * \sum_{i=1}^{N} log\pi(a_i = SKIP | a_{0...i-1}, b_{0,...,i-1})$$

For our implementation the budget objective is defined as follows.

$$O_{budget} = R^A * (num\_blocks * b\_w)$$

where num_blocks is the number of walls the adversary placed in this training step, b_w is a hyperparameter used to weight the budget and R^A is the reward of the antagonist. This budget objective is then added to the regret for the adversary.

IV. Setup

In the following we will outline on a structural level how we adapted the reward and goal statement, as well as explain what the environment is composed of and which techniques we employed in order to visualize our results.

A. Environment

The environments which are generated are 15 * 15 grid worlds. Therefore, the environment is discrete and 2D. The underspecified environment has free parameters concerning the position of the walls which is set by the distribution which corresponds to the environment policy and is approximated by the PAIRED algorithm. Thus, the adversary places the walls in the environment and also places the agent at a certain position. The agent cannot be set to the position of a wall. The edge of the grid is closed with wall tiles which can be seen in Figure 1.

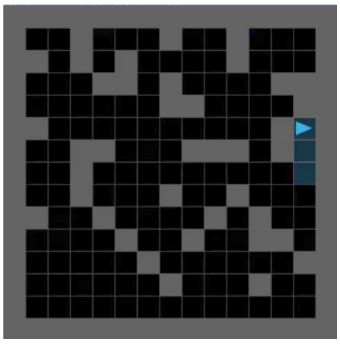


Fig. 1: Grid world generated by PAIRED.

B. Reward

[15] demonstrates the influence of different rewards on the performance for autonomic map generation. These findings inspired us when we had to define a reward. It was shown that a map completeness reward function achieved the fastest learning, whereas the information gain based reward made the agent learn more safe exploratory paths. The paper also reveals that if we reset the agent in order to penalize him for colliding with objects, the agent has to learn in the first place how to perfectly avoid collisions before he is able to navigate through the room. For mapping an unknown environment, early stopping is not beneficial and it increases the probability of getting stuck in local optimal policies, for example circling in one room.

Based on these results we decided to only give a penalty to the reward instead of resetting the agent when he collides with objects. Overall, we decided to give a positive reward for every new tile that was found in the map, as well as give a bigger positive reward for completing the map. For every time step we gave a small negative reward in order to force the agent to find something new. For colliding with tiles we gave bigger negative rewards in order to punish the agent accordingly. Formalized one can summarize that we used a map completeness reward, as well as we punished the agent concerning negative influence on the reward and on top of that introduced a continuously negative reward per time step. This is how we adapted the reward to our automatic map generation task. The trial terminates if a certain amount of time steps is reached, or if the map is explored completely.

C. Map generation

For the map generation we had to implement a detection if the map is completed, so that we ensure that the trial is terminated if the agent has explored the full map. Next to that we also coded an automatic map scaling in a way that if the agent is viewing something that would not be part of the map anymore, the map will be scaled bigger automatically. In our grid world example, this is not relevant, since the grid world is static and has a preselected size, but this is relevant for real world applications where the agent needs to adapt its map to the environment which does not have a fixed size. As a conclusion, this implementation may be used when transferring the results to a more complex or even real world environment. Moreover, it also demonstrates how important it is to stick the map to the environment more flexibly when it comes to more complex environments, as well as it is demonstrating that our selection of a small-scale, simple grid may lay the focus more on the implementation in the first place, and allow to analyze how exactly the algorithm behave in this certain circumstances in order to build a fundamental basis for further, more complex environments which can solve other robotic mapping issues like localization, noise, etc. We believe that these less computationally intensive grid worlds lend themselves well to a more critical and thorough analysis of the performance, behaviors, and intricacies of the algorithm. Another point which is also discussed by [7] is that the

traditional small-scale environments can still yield valuable scientific insights and can also help reduce the barriers to entry for underprivileged communities which do not have the opportunity to work with computationally complex environments.

D. Visualization

In order to visualize our results, we made use of a tensorboard. We extracted the data from Tensorboard in which we were interested in and used the programming language R to plot the data respectively to out needs. In the paper we included pictures of the grid, and pictures of statistics that are showing different results of performances. Because tensorboard does not support videos, we included .mov files of how the agent behaved in the environment in our GitHub repository here.

V. Results and Discussion

What we tested[1]:

- Normal PAIRED (R1)
- Flexible PAIRED
- Budget PAIRED
- Flexible and budget PAIRED
- Normal PAIRED without continuously negative reward (R2)

In order to evaluate the given performance of the different algorithms we decided to introduce certain metrics. First Findable Tiles, which is supposed to describe the number of tiles (no matter the object type) that can be found by the agent in this map. Second, we used Num Blocks which correspond to the number of walls (blocks) that are adversely placed during this training step. Tiles Found, that is the number of tiles the agent discovered this round (added to the map), was also logged. And lastly, Tiles To Map Completion which can be calculated by subtracting Tiles Found from Findable Tiles. It is important to note that Tiles Found does not by itself give a good picture if one version performs better than another, as it is possible that the environment is designed such that there are less Findable Tiles, which means that even perfectly performing agent could find less tiles in such an environment than a worse performing agent could find in an environment with a larger number of Findable Tiles. Tiles To Map Completion therefore provides a more accurate view, as it shows how close an agent gets to completing the full map of the environment. Generally, it is important to note that all results are averaged over multiple environments run in parallel. This explains why it is possible to for example get fractions of tiles for the Num Blocks metric.

On the one hand, we tested and compared the performance of the 4 different versions, namely PAIRED, budget PAIRED, flexible PAIRED and the combination flexible b-PAIRED. On the other hand, we also tested if the continuously negative reward (R1) has a significant influence on the performance and tested if the vanilla PAIRED will perform differently without the negative reward (R2).

A. PAIRED Variants

As can be observed in 2, the metrics Tiles Found can be evaluated as the following. Flexible b-PAIRED and PAIRED are increasing the most, whereas flexible PAIRED is increasing the least. All of the variants are increasing around 30-60 tiles, and they all start at 120-125 tiles. The maximum is reached by flexible b-paired with a value of 185. Consequently, it can be concluded that all of the algorithms seem to learn an underlying pattern of how to solve the given issue at hand and get better as the number of found tiles tends to increase, meaning that the agents explore more of the world.

The next figure (3) displays the values for the metric Tiles To Map Completion. Generally, it can be observed that there is a respective tendency for all graphs to decrease. Nevertheless, it looks like flexible b-PAIRED seems to perform the best

[1]the exact parameters we used for running the code can be found in the appendix

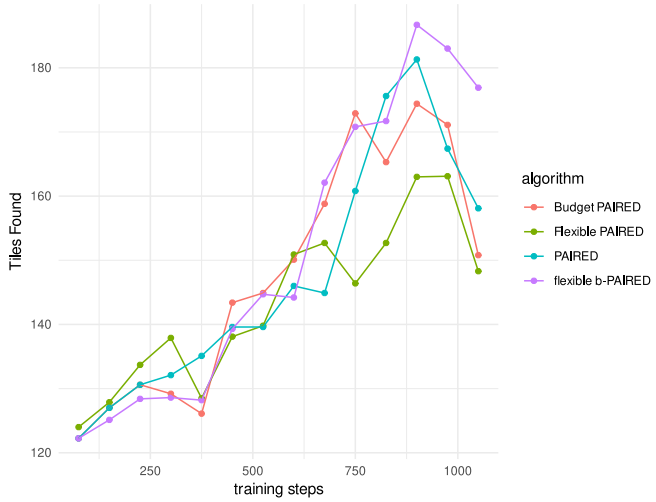


Fig. 2: Tiles Found

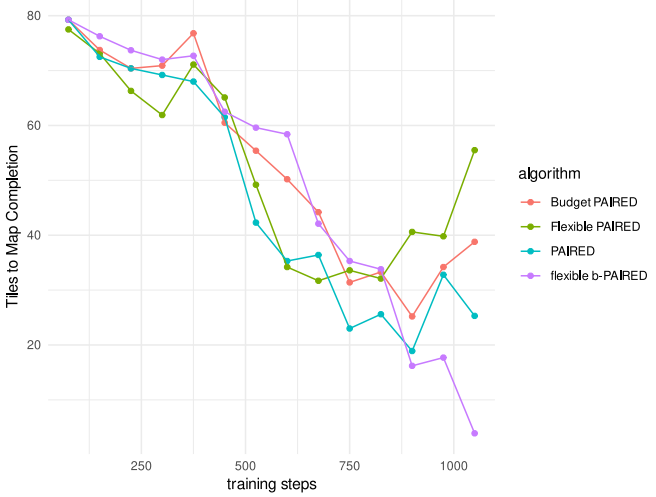


Fig. 3: Tiles To Map Completion

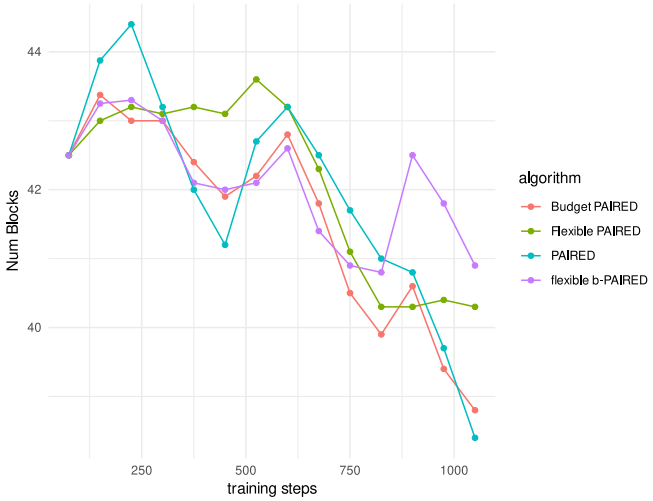


Fig. 4: Num Blocks

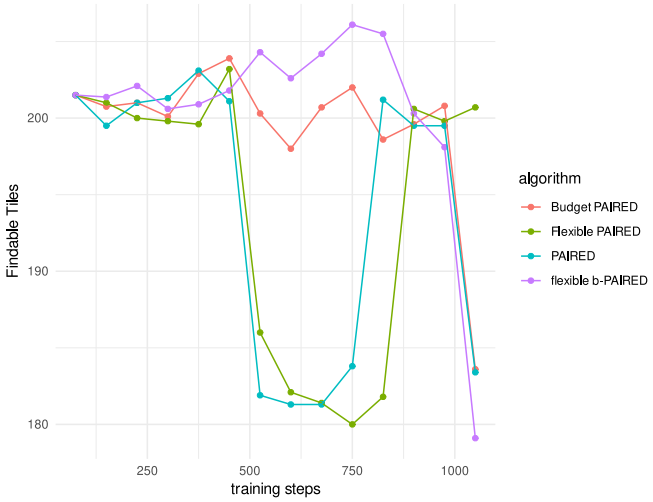
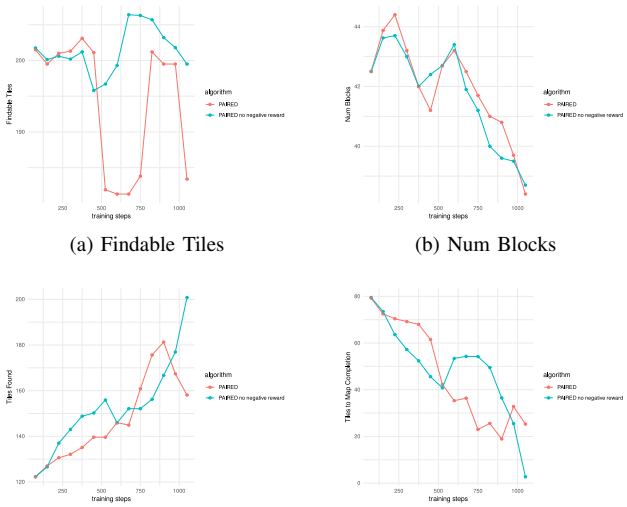


Fig. 5: Findable Tiles

since its graph decreases the most with almost no increase in between. B-PAIRED is followed by PAIRED and Budget PAIRED, whereas flexible PAIRED increases quite a lot after around 800 steps and is thus overall performing the worst. The difference of the number of tiles is emphasising here that flexible b-PAIRED performs in comparison really good, since for the algorithm it reaches 5 tiles for the last timestep, whereas for flexible PAIRED this is 55 tiles.

Moreover, the metric Num Blocks is shown in 4. Generally speaking, the metric has no especially striking features. Rather, the metric does not change much, only decreasing by around 2 blocks. What can be said is that it tends to decrease for all of the algorithm variants which might indicate that we did not train for long enough. We expected the graph to start with a lower block and to increase depending on whether the agent learned. We also assume that there was not enough time for the adversary to learn to reduce the number of blocks



(a) Findable Tiles

(b) Num Blocks

(c) Tiled Found

(d) Tiles To Map Completion

Fig. 6: Comparing performance of R1 and R2

and afterwards then adapt it according to the ability of the agents. Since the given paper trained for a lot longer, they had the resources to do so, and received better results. We inferred from that, that longer training might have improved our results in a way that they are showing more details about the performance and are more representable.

Lastly, the results for the metric Findable Tiles can be seen in 5. What seems to be striking is that PAIRED and flexible PAIRED both go down by around 20 tiles and afterwards show a tendency up again around 500 to 800 steps. We assume that this may have something to do with walls being placed in such a way that one part of the world is cut out, which is connected to the decrease of tiles, as we know that the number of walls (NumBlocks) does not increase, which would be the only or explanation for a decreasing number of Findable Tiles. Apart from that, the graphs are mostly staying around the values of 200 tiles. Flexible b-PAIRED is increasing to up to around 206 tiles which is the maximum thereof.

B. Comparing R1 with R2

Figure 6 shows the metrics for the two reward conditions. Just like when comparing the different algorithm variants, the metric Num Blocks does not change much (6a), but tends to decrease for both versions over all time steps. The difference between the maximum and minimum is around 5 blocks, starting around 42.5 blocks and decreasing to roughly 38.5 blocks. For both reward variants the pattern of change is similar, both increase and decrease around the same time steps.

As can be seen in 6b, Tiles Found tends to increase for both reward variants, however, it increases more in the case of R2. While both R1 and R" start at slightly over 120, the one R2 increases to slightly over 200, while R1 maximally reaches around 180 and decreases by over 20 tiles in the last 175 training steps to a value slightly below 160 tiles.

The next subfigure (6c) displays the metric Findable Tiles. As already described previously, for R1 the metric Findable Tiles stays around 200 until dropping by 20 tiles at 500 training steps, then increasing again at around 750 steps while decreasing another time at 1000 training steps. For R2 this metric looks differently, it only varies slightly around 200 tiles until training step 375, then decreases by around 5 tiles, before rising by 10 tiles over the next 250 steps and then decreasing to around 200 until the last training step.

Lastly, subfigure 6d shows the metric Tiles To Map Completion, which overall decreases for both reward versions. Starting at 80 it decreases to a minimum of just below 20 at roughly 930 steps for R1, while for R2 this metric decreases to a value of around 3. Strikingly, for R2 the metric increases by around 7 to a value around 55 tiles left to map completion between 500 and 600 training steps and plateaus there until decreasing again after step 750. This could somewhat be explained by the number of Findable Tiles also rising and falling around these training steps, but as the number of Findable Tiles does not vary by the same but rather by smaller amounts, this cannot explain the phenomenon fully.

VI. Conclusion

Overall, we tested how the PAIRED algorithm performs when it comes to training an agent in terms of automatic map generation. From our results we infer that the algorithm can be used for the specific field of robotic mapping as 2 reveals. In addition, it was clarified in the statistics, that the two improvements of the PAIRED algorithm, namely budget and flexible PAIRED, performed quite similar to the PAIRED algorithm and at some points even worse. But the flexible b-PAIRED algorithm has learned the most efficiently as could be observed.

Slight differences between the two reward versions where observable, with R2 seeming to perform slightly better with the the number of tiles found increasing more than for R1 towards the end of the training and Tiles To Map Completion also be lower at the last training steps. However, it is not clear whether these trends would continue, and so, as the differences between the rewards are not present during many training steps, we cannot draw any conclusions about the effectiveness of the different types of rewards and have to refer to further studies to test these for a longer training period.

In a broader context we introduced useful metrics to evaluate the performance of the algorithm concerning automated map generation. Next to that, we gave rise to a learned policy which can be transferred to more complex and large environments. Finally, also for further work we provided an approach how to generalise maps better, so the map size is adapted automatically. In our experiment this was not used because we employed Grid worlds, but for more complex environments this may be a sensible approach.

VII. Future work

In the first place we strengthen the emphasis on the need for more research in the field of automatic generation of robotic mapping solved with Deep reinforcement learning. As a first step it may be interesting to compare our performance with the performance of non-DRL approaches. Moreover, it may be interesting to see how the PAIRED-algorithm would work with different DRL algorithms since it can be combined with any DRL-algorithm. We would be interested in the combination of PAIRED and DRQN, as well as QMDPN as we saw these algorithms were already used for automatic generated mapping. Since we also highlighted in our work the importance of Adversarial Environment Generation in order to make successful learning of the agent's policy possible, it may be interesting to see how the PAIRED algorithm may improve the performance of these plain algorithms. Next to that, it may also be of interest to add more real world settings to our approach in order to see how our policy performs in real world applications. For example, introducing Noise or using ROS to simulate a real world application, could be a starting point here. As already mentioned, it would also be necessary to run the training for longer to be able to properly compare the effect of different rewards to automatic mapping in combination with PAIRED, at which point one could also test further reward versions, for example testing different weightings of the reward components.

References

[1] Dennis, M., Jaques, N., Vinitsky, E., Bayen, A., Russell, S., Critch, A., Levine, S. (2021). Emergent Complexity and Zero-shot Transfer via Unsupervised Environment Design. doi:arXiv:2012.02096.

[2] Gur,I., Jaques, N., Malta, K., Tiwari, M., Lee, H., Faust, A. (2020). ADVERSARIAL ENVIRONMENT GENERATION FOR LEARNING TO NAVIGATE THE WEB. doi:arXiv:2103.01991.

[3] Chen, G., Pan, L., Chen, Y., Xu, P., Wang, Z., Wu, P., Ji, J., Chen, X. (2020). Robot Navigation with Map-Based Deep Reinforcement Learning. doi: arXiv:2002.04349.

[4] Chen, F., Bai, S., Shan, T., EngloSelf, B. (2019). Learning Exploration and Mapping for Mobile Robots via Deep Reinforcement Learning. doi:http://dx.doi.org/10.2514/6.2019-0396.

[5] Kahn, G., Villaflor, A., Ding, B., Abbeel, P., Levine, S. (2018). Self-supervised Deep Reinforcement Learning with Generalized Computation Graphs for Robot Navigation. doi:arXiv:1709.10489.

[6] Karkus, P., Hsu, D., Lee, W. (2017). QMDP-Net: Deep Learning for Planning under Partial Observability. doi:arXiv:1703.06692.

[7] Osband, I., Doron, Y., Hessel, M., Aslanides, J., Sezener, E., Saraiva, A., McKinney, K., Lattimore, T., Szepesv´ari, C., Singh, S., Van Roy, B., Sutton, R., Silver, D., van Hasselt, H. (2020). Behaviour suite for reinforcement learning. In International Conference on Learning Representations. URL https://openreview.net/forum? id=rygf-kSYwH

[8] Obando-Ceron, J., Castro, P. (2021). Revisiting Rainbow: Promoting more Insightful and Inclusive Deep Reinforcement Learning Research. doi:arXiv:2011.14826

[9] Thorpe, C., Hebert, M., Kanade, T., Shafer, S. (1988). Vision and navigation for the Carnegie-Mellon Navlab. doi:10.1109/34.3900

[10] Urmson, C., et. al. (2008). Autonomous driving in urban environments: Boss and the urban challenge. doi:1395073.1395077

[11] She S., Michael, N., Kumar, V. (2011). Autonomous multi-floor indoor navigation with a computationally constrained MAV. doi:2011.5980357

[12] E. Olson. (2008). Robust and Efficient Robotic Mapping. Ph.D. dissertation, MIT

[13] Fuentes-Pacheco, J., Ruiz-Ascencio, J., Rendon-Mancha, J. (2015). Visual simultaneous localization and mapping: a survey. doi:s10462-012-9365-8

[14] Papadimitriou, C.,Tsitsiklis, J. (1987). The complexity of Markov decision processes. doi:moor.12.3.441

[15] Botteghi1, N., Sirmacek, B., Schulte, R., Poel, M., Brune, C. (2020). REINFORCEMENT LEARNING HELPS SLAM: LEARNING TO BUILD MAPS. doi: isprs-archives-XLIII-B4-2020-329-2020

[16] Garaffa, L., Basso, M., Konzen, A., de Freitas, E. (2021). "Reinforcement Learning for Mobile Robotics Exploration: A Survey," IEEE Transactions on Neural Networks and Learning Systems, pp. 1–15. doi: 10.1109/TNNLS.2021.3124466

[17] Zhu, D., Li, T., Ho, D., Wang, C., Meng, M. (2018). "Deep Reinforcement Learning Supervised Autonomous Exploration in Office Environments," IEEE International Conference on Robotics and Automation (ICRA), May 2018, pp. 7548–7555. doi:10.1109/ICRA.2018.8463213

APPENDIX

A. *Run parameters*

What used the following parameters for running the code:

- Normal PAIRED

```
--debug --root_dir=tmp/results_1/ --random_seed=42 --flexible_protagonist=False
--block_budget_weight=0 --num_train_steps=600
```

- Flexible PAIRED

```
--debug --root_dir=tmp/r1_1/ --random_seed=42 --flexible_protagonist=True
--block_budget_weight=0 --num_train_steps=600
```

- Budget PAIRED

```
--debug --root_dir=tmp/r1/ --random_seed=42 --flexible_protagonist=False
--block_budget_weight=0.1 --num_train_steps=600
```

- Flexible and budget PAIRED

```
--debug --root_dir=tmp/r4/ --random_seed=42 --flexible_protagonist=True
--block_budget_weight=0.1 --num_train_steps=600
```