Hindsight Experience Replay with Environment Relabeling

Final Report

Katharina Hermann
Technische Universität München
Department of Informatics

Ferenc Török
Technische Universität München
Department of Informatics

***Abstract*—This document is written as a final report for the project of the course Advanced Deep Learning in Robotics at the Technische Universität München. Our chosen topic is to develop a method to train a Neural Motion Planner (NMP) [1] agent with Reinforcement Learning (RL) [2] in an environment with randomly placed obstacles. The aim of the method is to reduce training time as much as possible meanwhile using a very simple and sparse reward function. For this we use Hindsight Experience Replay (HER) [3], [4] but we extend it by not just assigning a new goal, but also a new workspace to the unsuccessful trajectories.**

I. Experiment Environment Setup

A. Agent

We have used a 2D point-mass robot with a given radius for the sake of simplicity. If an obstacle is closer to the robot than its radius, a collision occurred. Also, if the goal is within the agent's radius, it is considered to be reached. The agent has a continuous action $a_t = [a_{x,t}, a_{y,t}]^T$ (x and y direction step distance) with which it is able to interact with the environment. The norm of the action vector is constrained to be unity distance, that is: $||n_t||_2 = 1 \quad \forall t$.

B. Reward specification

The agent's task is to navigate from the start point to the goal without colliding in a workspace (WS) with obstacles. Our reward function r_t is as simple as it gets: the agent receives $r_t = +8$ if it has reached the goal, $r_t = -7$ if it has collided with an obstacle or the WS wall and $r_t = -0.06$ after every timestep. The optimal rewards have been tuned using random search.

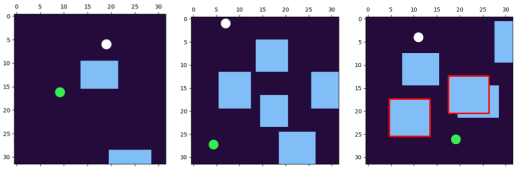
C. Workspace

Workspaces are represented as a 32×32 grid, where each grid cell is either free space (value 0) or obstacle (value1). For training we use a WS, with $4-5$ obstacles (randomly chosen). Their position is generated with a uniform and the size with a normal distribution with mean $8cm$ and variance $2cm$. For testing we use 3 different WS levels: easy, medium and hard. The easy WS (Fig:1(a)) contains $2-3$ obstacles with a mean size of $8cm$. The medium WS (Fig:1(b)) is the same as the one we used for training, except that the obstacles do not overlap. The hard level WS (Fig:1(c)) contains $5-6$ obstacles with mean size 8, whereby 2 of them always form a narrow passage between start and goal. Examples of the different difficulty levels are illustrated in Figure 1.

Start, goal: Start (Figure 1, white point) and goal (Figure 1, green point) positions are generated randomly for a given WS.

D. Performance metric

We use success rate as the evaluation metric. In every evaluation cycle, we carry out N evaluation episodes. The succes rate for an evaluation cycle is then calculated as follows: $\mu = N_{succ}/N$, where N_{succ} is the number of successful episodes. In addition, to test, whether the agent just learns to find goals, where a straight-line trajectory is feasible, or whether it finds, more challenging goals, we check the percentage of WS with a a feasible straight-line trajectory: $\mu = N_{straight-line}/N$. We then calculate the success rate for the straight-line WS and the more complex WS separately.



(a) Easy WS (b) Medium WS (c) Hard WS

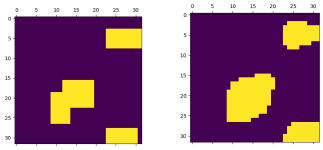
Fig. 1. 3 workspace levels with obstacles (light blue), agent (white) and goal (green)

II. Training Architecture

For the environment we have implemented a custom OpenAI gym environment (Left side of Fig. 2). We have chosen the agent to be trained with Deep Deterministic Policy Gradient (DDPG) [5] an off-policy actor-critic algorithm for continuous state and action spaces (Right side of Fig. 2) .

The state of the agent s_t in the RL setup contains its position $c_t = [x_t, y_t]^T$, the goal position $g = [g_x, g_y]^T$ and the workspace in a latent representation w: $s_t = c_t, g, w$ The workspace in its original form is represented with a feature vector of size 1024. Compared to this, the state of the robot and the goal are only represented with $2-2$ features. Therefore we have implemented a Convolutional Autoencoder (CAE) for reducing the workspace representation size in order not

to lose the information about the state and the goal position. We trained and tested the CAE on 10000 workspaces. For the encoder we have used 3 Convolutional layers with filter sizes of $[4, 8, 16]$ followed by max-pooling for size reduction. Then after a linear layer the feature size was reduced to the latent space dimension, which is 16. For the decoding we have used a linear layer followed by transpose convolutional and convolutional layers. We have used weighted binary cross entropy loss for training. The accuracy on the test set is 96%. Figure 3 shows an example of the original and the output image.



(a) Input (b) Output

Fig. 2. The input and output of the trained Convolutional Autoencoder on a test image.

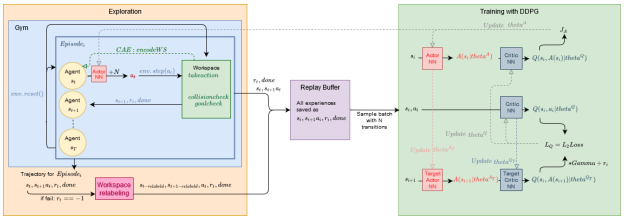


Fig. 3. The DeepRL setup for training the agent. Left upper part: Exploration with the Gym implementation, Left lower part: Goal and WS Relabeling based on HER, Middle: Replay Buffer, Right: Training according to the DDPG approach, with 4 NN (Actor, Target Actor, Critic, Target Critic)

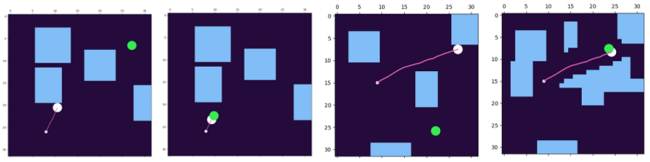
III. Relabeling Strategies

We have designed 2 different relabeling strategies: ”Cutting trajectory relabeling” and ”Random passage relabeling” (Fig. 4)

The method for ”Cutting trajectory relabeling” removes the last part of the trajectory, when the robot collided with an obstacle. If the agent has hit the WS wall, the obstacles and the trajectory are shifted from the wall. The new goal in both cases is placed around the last state of the trajectory. Care has been taken, to place the new goal so, that it is only considered to be reached from the last state. If the agent has only reached the max number of steps, we only replace the goal. Some examples can be seen in Figure (4).

The method for ”Random passage relabeling” works similar to ”Cutting trajectory relabeling”, except that for each relabeled trajectories new obstacles are generated, in a way, that they surround the trajectory and form a narrow passage as soon as the trajectory, gets straight.

Most of the trajectories during warmup and some trajectories during training, are far from being smooth, they zig-zag as the agent jitters (Fig. 5). This can be a problem, as suboptimal trajectories are relabeled and saved in RB as perfect“ samples.



(a) Before (b) After (c) Before (d) After

Fig. 4. The two relabeling strategies exemplary for the training case, when the agent has hit a goal. a) and b): ”Cutting trajectory relabeling”, c) and d) ”Random passage relabeling”

Therefore, the robot will possibly not learn to avoid jittering movement. To test this hypothesis, we also carry out relabeling trainings, where we drop the trajectories from relabeling which were zig-zagging too much.

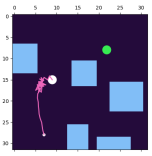


Fig. 5. An example for a zig-zagging trajectory

IV. Training and Evaluation Setup

We trained and evaluated the agent on 3 different WS setups. We performed 800k training steps with a policy update after each step. The warmup phase has been chosen to be 10k steps. The batch size for samples from the replay buffer has been chosen to be 265. The following hyperparameters have been optimized with random search: rewards, learning rates of actor and critic network, sigma (parameter for randomness in action for exploration.), tau (update rate of the target network weights), maximum allowed gradient (gradient clipping). We used weight decay of 0.001 as regularization. For training and evaluation 2 different WS buffers of 100 WS have been generated, from which the workspaces have been sampled for each episode. We tested the agent for 1000 episodes in previously unseen workspaces.

A. Empty WS

To verify the implementation and to check, whether we can reach a good baseline, we first trained and evaluated the agent on an empty WS with 4 different training strategies: ”Straight-line relabeling” is not an actual relabeling method, but adapts the strategy for solving NMP problem by saving perfect trajectories into the replay buffer for unsolved scenarios [1]. In the case of an empty trajectory the perfect solution would be a straight line between the start and the goal. Accordingly

1	No relabeling
2	”Cutting trajectory relabeling” with removing zig-zag trajectories
3	”Cutting trajectory relabeling” without removing zig-zag trajectories
4	”Straight-line relabeling”

in this case we do not relabel the trajectory, but instead save a straight trajectory as an optimal solution. By relabeling the trajectories always with the optimal straight-line trajectory, we expect to reach 100% accuracy very fast.

B. Medium WS

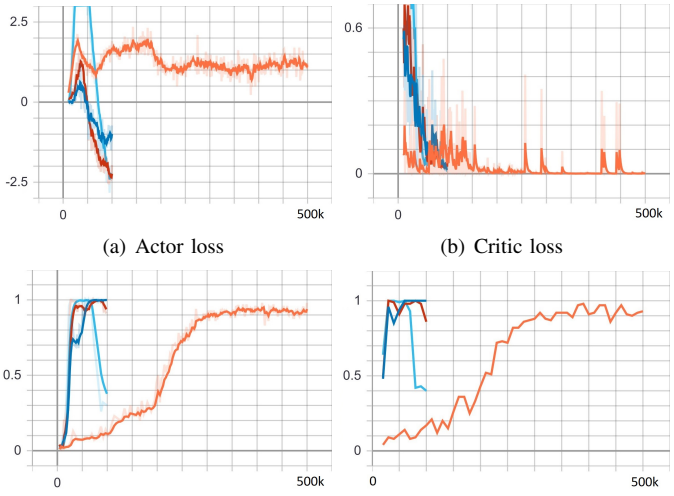
The actual training in a WS with obstacles is performed in medium difficulty workspaces as stated in I-C. We train with 5 different strategies:

1	No relabeling
2	"Cutting trajectory relabeling" without removing zig-zag trajectories
3	"Cutting trajectory relabeling" with removing zig-zag trajectories
4	"Random passage relabeling" without removing zig-zag trajectories
5	"Random passage relabeling " with removing zig-zag trajectories.

For testing, we evaluate the 5 agents in the 3 different WS levels.

V. Results

A. Empty WS Training and Evaluation



(a) Actor loss (b) Critic loss

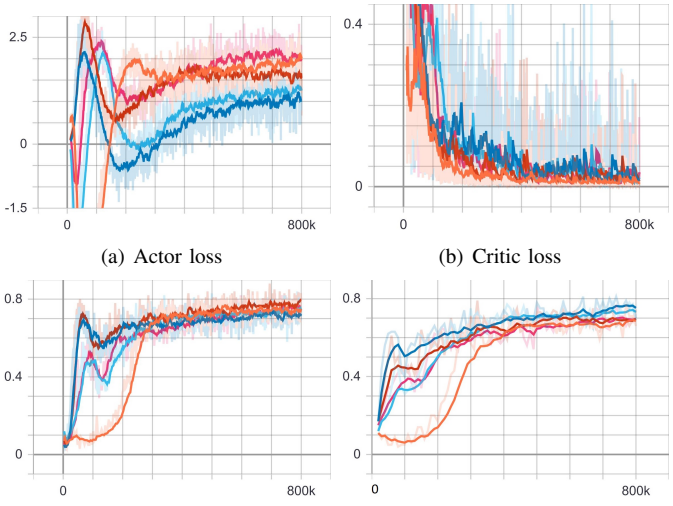
(c) Training accuracy (d) Validation accuracy

Fig. 6. Sanity check in empty workspace. Orange: no relabeling, dark blue: cutting trajectory without zig-zag removal, red: cutting trajectory with zig-zag removal, light blue: straight line

B. Medium WS Training

C. Medium WS Evaluation

The 3 tables I, II, III show the evaluation results on all 3 WS levels. Beyond the total success rate of the agent, 2 other metrics were also used: the success rates in scenarios which could have been solved by straight line motion and in scenarios which were more difficult.



(a) Actor loss (b) Critic loss

(c) Training accuracy (d) Validation accuracy

Fig. 7. Training in medium level WS. Orange: no relabeling, dark blue: cutting trajectory without zig-zag removal, red: cutting trajectory with zig-zag removal, light blue: random passage without zig-zag removal, pink: random passage with zig-zag removal

TABLE I
TEST ACCURACY ON EASY LEVEL WORKSPACES

Relabeling Method	**Straight line (SL) Feasibility**	**Test accuracy**		
		Total	**SL**	**No SL**
No relabeling	89%	88%	91%	62%
CT-No-ZZ	90%	96%	99%	70%
CT-ZZ	87%	95%	99%	65%
RP-No-ZZ	89%	94%	98%	61%
RP-ZZ	87%	90%	95%	60%

TABLE II
TEST ACCURACY ON MEDIUM LEVEL WORKSPACES

Relabeling Method	**Straight line (SL) Feasibility**	**Test accuracy**		
		Total	**SL**	**No SL**
No relabeling	55%	62%	91%	25%
CT-No-ZZ	58%	64%	93%	23%
CT-ZZ	54%	63%	94%	27%
RP-No-ZZ	59%	70%	97%	31%
RP-ZZ	59%	67%	93%	30%

TABLE III
TEST ACCURACY ON HARD LEVEL WORKSPACES

Relabeling Method	**Straight line (SL) Feasibility**	**Test accuracy**		
		Total	**SL**	**No SL**
No relabeling	1%	31%	100%	30%
CT-No-ZZ	1%	32%	75%	32%
CT-ZZ	1%	34%	100%	34%
RP-No-ZZ	1%	40%	100%	40%
RP-ZZ	1%	38%	80%	38%

VI. Conclusion

In summary the evaluation shows, that both relabeling methods with and without relabeling zig-zag trajectories result in a slight improvement of the agent's accuracy (about 10%).

Moreover with relabeling, the agent learns much faster than without relabeling.

However, in comparison to the strategy of [1] with expert trajectories our relabeling has not achieved equally good results. With their approach they achieved 0.9733 accuracy on an experiment with a robotic arm in a 3D workspace.

The advantage of our approach is, that relabeling the workspace is easier and computationally more efficient than calculating feasible solutions for difficult scenarios. However, simplicity always comes with some cost: We can only relabel the unsuccessful trajectories of the agent. If the agent only comes up with almost straight-line trajectories, we will not be able to generate hindsight trajectories which mimic good solutions for difficult scenarios. We lose the diversity of trajectories, on which we can train. According to this, in cases where a straight-line (or almost straight line) solution would be feasible our agents perform really well. However, the robot can not solve the scenarios for which more creative solutions would be necessary.

As an outlook, future work could investigate creating longer not straight but smooth trajectories in an empty workspace and relabeling them by placing obstacles around them. In this way it would be possible to come up with a lot of good examples before the training starts. Another approach could be to bend the trajectories during relabeling.

REFERENCES

[1] T. Jurgenson and A. Tamar, "Harnessing reinforcement learning for neural motion planning," *arXiv preprint arXiv:1906.00214*, 2019.

[2] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," 2011.

[3] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in neural information processing systems*, 2017, pp. 5048–5058.

[4] A. C. Li, L. Pinto, and P. Abbeel, "Generalized hindsight for reinforcement learning," *arXiv preprint arXiv:2002.11708*, 2020.

[5] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.