Deep Robust Navigation with Cognitive Mapping Visual Representations

Titouan Renard (MT-RO, 272257), Umer Hasan (IN-H, 347099), Yongtao Wu (IN-H, 346107)

Presentation Video
Habitat-Lab implementation repository
Gym-Miniworld implementation repository

Abstract

In the following article, we discuss a novel deep-learning method for autonomous navigation that we call Deep Robust Robot Navigation (DRRN). The core of the idea is to enhance the robustness of deep cognitive mapping methods with mid-level visual priors. We go through the design choices that lead to DRRN and discuss implementation details and experimental results.

1 Introduction

The navigation problem is one of the cornerstones of mobile robotics and is still an open problem. Nowadays the field is split between classical geometric approaches (mostly SLAM-based) and machine-learning (mostly reinforcement learning) based methods. The former tends to be computationally quite expensive (at runtime) and lacks the ability to infer non geometric properties about the environment that humans are able to deduce without trouble. The latter has the ability to learn such properties and can be computationally much more efficient (at runtime) but policies are often quite brittle (small variations from training to testing conditions can have drastic impact on performance) and require long training. For this reason investigation into the robustness of such methods is an active research field.

This robustness limitation is the motivation behind the method that we propose below, since mid-level visual representations (MLReps) have been shown to improve robustness, it is quite natural to try apply them in conjunction with a state of the art visual-based navigation method such as deep cognitive mapping.

2 Related Work

Representation learning describes extracting useful features from the given inputs. Well-known algorithms such as independent component analysis and autoencoders are unsupervised while dictionary learning is supervised [2, 5, 4]. Task-relevant approaches such as MLReps first enable the encoder to learn the representation by designing different sub-tasks [16]. The generalized representation can be used directly for other tasks. Using MLReps also has a cognitive psychology motivation, which states that biological organisms obtain useful abstractions from visual information [12]. Visual priors have been shown to make reinforcement learning policies more robust. They facilitate sim2sim and sim2real transfer by isolating the relevant features from the environment. The notion of cognitive mapping also has it's roots in biology and psychology and was first explored in the mid-twentieth century[1].

Navigation has been investigated for a long time in robotics and computer science. Given perceptual information from the environment, traditional methods will first map these features into an explicit representation of the environment and then apply some kind of planning algorithm on that representation. One key research area in modern mobile robotics is the so-called simultaneous localization and mapping problem (SLAM). State of the art SLAM algorithms such as LSD-SLAM [7] extract key-points from monocular vision data, and use geometric techniques to explicitly compute a representation of the environment. That kind of representation can the be used by a planning algortihm (such as rapidly exploring random trees [3]) to compute a path for the agent to follow.

With the development of deep neural network and reinforcement learning (RL), plenty of end-to-end methods are proposed [9, 10]. On the other hand, [11] develops a cognitive mapping algorithm to provide a strong inductive bias on spatial transformations that has been shown to outperform naive approaches to point navigation with deep learning. The idea is to impose spatial translations to the "map" embedding to facilitate the learning of a persistent map representation of the environment. This gives a strict notion of memory to the agent.

3 Method

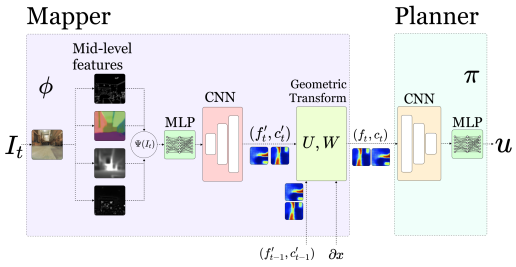


Fig. 1: Network architecture of the cognitive mapper network. From left to right : the image I_t is converted to a midlevel representation Ψ which passes through a MLP and a CNN to be decoded into a map update (f'_t, c'_t). This map update is then combined with the map at the previous time step (f'_{t-1}, c'_{t-1}). The combination is performed by the U and W functions that implement geometric transformations and map combination according to confidence. This gives a map representation (f_t, c_t) which is used by the planner π to compute an action distribution u.

3.1 Network architecture

As in [11] we decompose the navigation task into a mapper function ϕ that learns a world representation, and a planner policy π that controls the agent according to the representation (f, c) encoded by ϕ. The f channel represents obstacle free space in the world and the c channel encodes confidence. A schematic representation of our architecture can be seen in Figure 1.

3.1.1 Mapper Architecture: First, we describe our mapper function ϕ that at any time t generates a two channel cognitive map (f_t, c_t) (where f_t stands for free space map and c_t for confidence map). We propose to use mid-level representations instead of raw image pixels as input to our sensory system. We consider a set $\Psi = \{\psi_1, ..., \psi_m\}$ of functions that take an image I to a lower dimensional representation $\psi_i(I)$. We denote $\Psi(I)$ the set of representation of an image I. Our approach makes use of pre-trained representations from the taskonomy model set [16]. The representation encoders are fixed and not trained during the procedure.

At each time step t the set Ψ of representations is passed through a multi layer fully connected network and decoded to a dual channel (f'_t, c'_t) map representation update. Two functions, U and W combine the map representation update (f'_t, c'_t) with the map representation from the previous time step, denoted (f_{t-1}, c_{t-1}). The W function implements an affine transform that ensures that the prediction at time $t-1$ is overlayed with the prediction at time t for a given egomotion $\partial x = [\partial x, \partial y, \partial\theta]^T$. This is performed by rotating and translating (f_{t-1}, c_{t-1}) in such a way that the agent is placed in the center point of the map. W is fully differentiable [8]. We implement the following affine transform matrix $T_{\partial x}$:

$$T_{\partial x} = \begin{bmatrix} \cos\partial\theta & \sin\partial\theta & -\cos\partial\theta\partial x - \sin\partial\theta\partial y \\ -\sin\partial\theta & \cos\partial\theta & -\sin\partial\theta\partial x - \cos\partial\theta\partial y \\ 0 & 0 & 1 \end{bmatrix}. \quad (1)$$

The U function combines the map representation update (f'_t, c'_t) with the transformed map representation at time $t-1$ according to the following rule:

$$\frac{f_{t-1}c_{t-1} + f'_t c'_t}{c_{t-1} + c'_t}, \qquad c_t = c_{t-1} + c'_t. \quad (2)$$

The U function outputs the map representation at time t, denoted (f_t, c_t).

3.1.2 Planner Architecture: We use a simple planner architecture taking the map representation (f_t, c_t) as input. The representation is first fed into a series of 2D convolutional neural networks to capture the geometric information and reduce the dimensionality. The output then goes through a few dense layers that represent an action distribution. Such an architecture is the standard for pixel-to-action reinforcement learning (RL) [18]. We learn over a discrete action space (such as "`turn_left`, `turn_right`, `forward`" although the exact discrete action set varies across our implementations).

3.1.3 Training: Since the entire network (ϕ and π) is differentiable we choose to train it using a policy gradient (PG) based RL algorithm, i.e., proximal policy optimization (PPO) [13]. Compared with more naive policy gradient methods, PPO leads to relatively fast convergence for a given amount of data in storage while being relatively robust to reward collapse. Altough it's computational complexity is relatively high, it tends to lead to some of the fastest "wall-time" convergence. It is an actor critic method that shares some benefits with trust region methods such as TRPO. The actor network outputs an action distribution while the critic network outputs an advantage function which gives an approximation of a state's value. It is worth mentioning that the choice of training a neural network planner together with the mapper network through policy gradient diverges from the approach taken in [11] where the planner is trained fully supervised using DAGGER [6] and an expert policy.

Training with RL implies formulating the problem as Markov decision process (MDP), or more accurately as a partially observable Markov decsision process (POMDP) where we associate state transitions with a reward function. First, consider the sparse reward function :

$$r_{\text{sparse}}(s,a) = \begin{cases} \frac{1}{t} & \text{if } |x - x_{\text{goal}}| < \epsilon \\ 0 & \text{else} \end{cases}, \quad (3)$$

where x denotes the agent's position x_g the goal position and t is the time since the beginning of the episode. This reward function can lead to converging policies although convergence is slow, as the trainig algorithm gets little feedback per run. Another approach is to give a reward whenever the agent approaches the goal, this gives an incentive to explore which yields a faster convergence. We express that second reward as :

$$r_{\text{shaped}}(s,a) = \begin{cases} \frac{a}{t} & \text{if } d_t < \epsilon \\ b & \text{if } d_t < d_{t+1} \\ -b & \text{if } d_t \geq d_{t+1} \end{cases}, \quad (4)$$

where d_t is the distance of the agent to the goal at time t, a and b are tuning parameters that are associated with a specific implementation.

4 Experiments and Results

4.1 Experimental Design

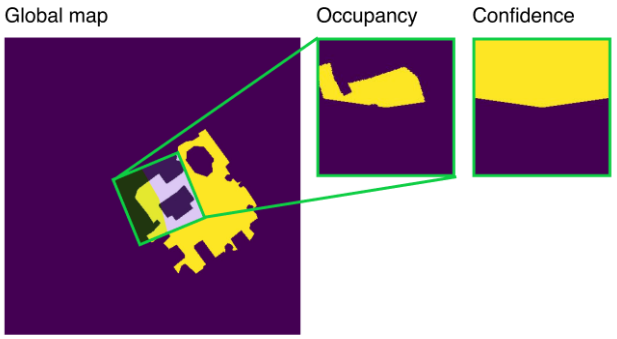
4.1.1 Habitat Environment [17]: We implement most of our code by extending the `Sensor` class available in the Habitat sim library. This worked well with the existing Habitat code since each sensor is a separate modular component that can be added to experiments by specifying the sensor in the configuration file. This design choice ensured the code was kept modular as all additional inputs to the RL policy were in the form of a separate Habitat sensor. Each sensor class overrided the `get_observation()` function to return the correct tensor. In particular we implemented:

1. `HabitatSimMidLevelSensor` - uses the result of `HabitatSimRGBSensor` (the current RGB image the agent sees) and converts it to the specified MLReps using a helper function which calls the `visualpriors` package. The package contains pre-trained models that return MLReps.

2. `AgentPositionSensor` - calculates the displacement of the robot from the previous observation by using the current position and storing the previous position. The displacements have to be converted using affine transforms to be relative to the agent instead of the environment.

3. `HabitatSimMidLevelMapSensor` - holds the map generated from the mid-level observations and the egomotion observation. This is the critical part of the code for the DRRN policy. We employ a useful trick here by using the sensor as a means of storing the map generated by the mapper component of the architecture. Originally, we planned to hold the map within the `rnn_hidden_states` of the PPO policy, however, this led to implementation bugs in the code. Instead we use this sensor to return the previous map that was calculated with the mapper. Specifically, the sensor
(a) decodes a map representation from the mid-level representation
(b) applies the egomotion transform to update the previous map (the W function)
(c) creates a new map using the update equation defined above (the U function)

Fig. 2: We compute a topdown map from the global map and use a fixed confidence to initialize the PPO policy with accurate map information.



(d) stores the new map to be returned in the next observation and returns the previous map that was already stored

4. `HabitatSimMapSensor` - holds the actual map relative to the agent's position. This was quite an involved section of the code. We first get the global map of the environment and cache it in a variable (to avoid doing extra unnecessary work every time the agent moves). We calculate the displacement vector to apply to the map by transforming the world displacement we get from the simulator. In order to apply the affine transformations efficiently, we make use of cupy (cuda enabled library) and apply them in one transformation. Once we have the map representation relative to the agent, we apply a 90° cone of vision to only show the agent the map in front of it. We also include a confidence channel, which is 100% confidence for the cone of vision and 0% otherwise. (see figure 2)

Additionally, we would like to add that we implemented a model for supervised training of the mapper in `supervised_training.py`. Unfortunately, we did not run experiments with this approach. This approach was designed to mitigate the unstable gradients that are inherent to the PPO policy. The mapper would be trained to generate maps on a dataset of top down maps from the Habitat environment. The PPO policy would then take in a representation of the maps instead of a custom representation the policy decides on.

4.1.2 Miniworld Environment [15]: Since we implemented the policy for the habitat environment first, we ported most of the code from the previous implementation. For now, we duplicated the mapper architecture and made some changes to run it with the new environment. This made debugging easier but increased code complexity. We implemented each custom policy as a base policy class which was used by the `Policy` class. Namely, the 3 classes we implemented were `RGBBase`, `MidLevelBase`, and `DeepCognitiveMapper`. The functionality is similar to above.

During the experiments, we investigated how changes to the implementation of the reward function influenced the rate of convergence.

We calculate the euclidean distance between the agent and the goal and compare the current distance with the previous distance. If the agent gets closer to the agent, it will receive reward 0.2, otherwise 0. The reward for reaching the goal is 1.

In Table 1, we list the full architectures for the actor of PPO in miniworld (similar to the one in Habitat). The critic network is simply implemented by a fully-connected layer. Example observations of both environment are shown in Figure 3.

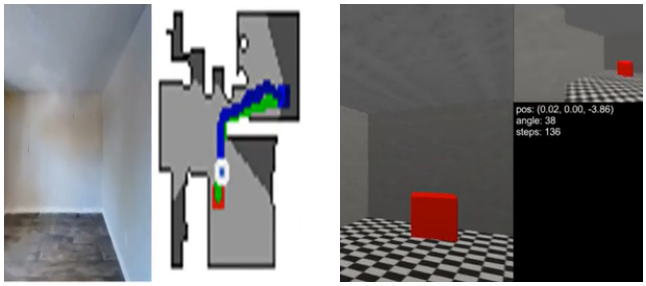
4.2 Evaluation metrics

1. `Success rate (SR)`. An episode is considered as success if and only if the final location of the agent is within 0.2m of the target position. Success rate is defined as the successful ratio of episodes.

Table 1 Architecture for the actor of PPO on Miniworld. Input map: (Batch,80,60,2).

Layer name	Kernel	Stride	Output channel
Conv2d	$(5,5)$	$(1,1)$	4
BatchNorm, ReLU	–	–	4
Conv2d	$(7,7)$	$(1,1)$	4
BatchNorm, ReLU	–	–	4
Conv2d	$(7,7)$	$(1,1)$	4
BatchNorm, Flatten	–	–	1024
Linear	(1024,128)	–	128
Tanh	–	–	128

Fig. 3: Example observations of the Habitat environment and Miniworld environment.



(a): Habitat environment (b): Miniworld environment

2. `Success weighted by Path Length (SPL)` [14]. Given the geodesic distance of the shortest path d_1, the total distance that the agent traverses in the current episode d_2, and the binary indicator of success s; SPL is expressed as $s * d_1/max(d_1; d_2)$, which measures the efficiency of reaching the target.

3. `Distance To Goal in meters (DTS)`. This metric is defined as the distance between the agent and the success threshold boundary at the end of an episode.

4.3 Results

4.3.1 Habitat Environment: The rewards during training and the metrics of our methods are plotted in Figure 4 and Figure 5, where we can see that our method didn't converge due to insufficient wall-clock time and lack of parameter tuning.

For the comparison with baseline, we were able to test 4 policies, however, due to time and compute constraints we were not able to run them for the same number of updates. It also seems we ran none of them until full convergence was achieved. The quantitative results are summarized in Table 2. The mid-level method achieves best performance in all metrics. Our DRRN outperforms the RGB sensor in terms of DTS, Success rate and SPL while the training speed of DRRN is slower in terms of NPS. As expected, the result of using the actual map is the upper bound of our method.

4.3.2 Miniworld Environment: We choose the RGB policy and the Mid-level policy as the baseline. The results are summarized in Table 3. In the Miniworld environment, we are able to train RGB successfully in a more complex map such as Y-maze. However, the training of DRRN and Mid-level fails. This will be deferred to future work. It is worth noting that the training of the RGB method is much faster than Mid-level and DRRN, both of which require computing the mid-level representation.

Table 2 Quantitative result on point navigation task on habitat dataset. NPS represents the total numbers of updates (units are 1000s). Mid-level achieves the best result in all metrics.

Method	#NPS	DTS (↓)	SR (↑)	SPL (↑)	Reward (↑)
RGB	640	5.017	0.2945	0.1545	0.7136
Mid-level	49	**3.369**	**0.5833**	**0.2584**	**5.982**
Actual Map	464	3.640	0.4261	0.2301	2.789
DRRN	177	5.28	0.3617	0.1930	1.918

Fig. 4: Metrics of DRRN during training

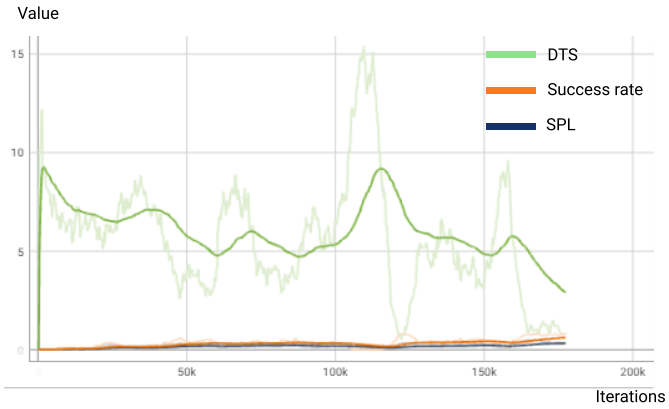


Fig. 5: Rewards of DRRN during training

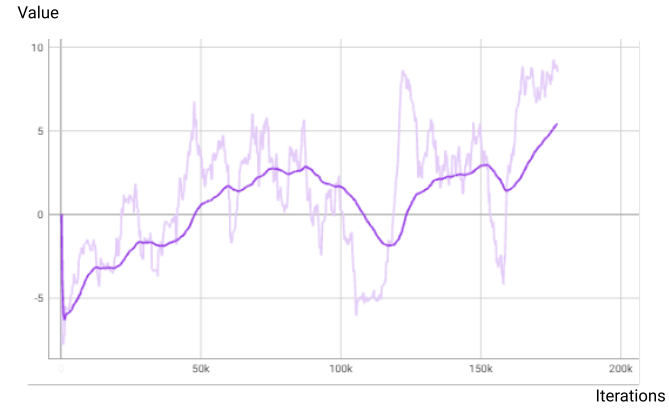


Table 3 Quantitative results on object navigation task in the Y-Maze map in the gym-miniworld environment.

Method	Frame per second	Success rate
RGB	507	**0.42**
Mid-level	52	0.12
DRRN	43	0.02

5 Conclusion

5.1 Limitations

Due to the time constraints of the project, we raise here a few limitations that we think are significant to take into account alongside the current results.

The experiments in their current state do not give much evidence towards whether our current policy outperforms the mid-level method or the RGB to cognitive map method. The experiments should be standardised to run for the same number of updates, they should run on the same training maps for the same amount of episodes, and they should also be evaluated on the same validation maps. Additionally, there is an option of uploading the policy to the Habitat submission server and seeing how it performs on the test set for the Habitat challenge.

The reasons for adding MLReps is to converge faster, generalise better, and train a more robust agent. The added value mid-level representations potentially add for generating cognitive maps has not been investigated fully. To address, this issue, we propose to run another baseline experiment which uses RGB input to build a cognitive map (the same approach taken in the cognitive mapping paper [11]). This could then be directly compared with DRRN and the difference would be due to the MLReps. This would reaffirm the assumption that the combination of MLReps and cognitive maps outperform the approaches when they are applied individually.

Note that at the moment we were only able to use 1 or 2 MLReps since they take a very long time to compute. To explore the full added value of MLReps, we would need to experiment with a combination of up to all 16 MLReps. The time complexity could be addressed by either simplifying the mid-level encoders through fine-tuning and making them compatible with smaller image sizes or black and white images, or by more compute.

Another limitation of MLReps is the fact that they make use of really deep resnets that take a long time to evaluate. This makes them unfit for decision making at high frequencies and reduce the possibilites of applications in robotics (for example in the case of a drone where policies need to run in a range of around $100Hz$ MLReps are not approriate). This limits the use-cases of DRRN to applications that are either not time-critical (either on a slow robot or in simulation) or that have a really high amount of compute available.

An interesting comparison would be between the end-to-end training of the DRRN policy, versus the DRRN policy with supervised training of the mapper. Our assumption is the latter would converge faster. It may or may not perform better depending on our inductive bias where we assume that the policy will learn best if supplied a map of the current surroundings. If trained for long enough, the end-to-end DRRN policy may come up with an even better internal representation to use.

5.2 Future Work

We suggest experimenting with our policy in different simulator environments and real-world environments.

In our proposal, we discussed converting this problem from 2D agent navigation (i.e. robot) to 3D agent navigation (i.e. drone). Although, we did not get around to implementing this, we believe it will be a very useful application of both of these state of the art techniques. A caveat is that it will involve extending our U and W functions for 3D transformations which will be quite an involved process.

Furthermore exploring a more expressive transformation function than affine transforms under perfect odometry assumptions would be of great interest. One could learn a transform from actual sensory measurements from a robot rather than directly impose it as in our current implementation.

Learning mid-level representations on smaller networks would also be of great interest as one of the limitations of DRRN comes from the computational complexity of computing MLReps from an

input image. It would therefore be of great interest to develop a lighter implementation of MLReps for applications in robotics.

6 Individual Contributions

U.H and Y.W. setup the habitat environment. T.R. setup the miniworld environment. U.H. and T.R. implemented the habitat sensor classes. T.R. and Y.W implemented the miniworld base policies. All members ran several experiments on google cloud VM instances. Y.W. setup ResNet decoder and supervised training network. T.R. implemented the W function. U.H. implemented the U function.

References

[1]Edward C. Tolman. "Cognitive maps in rats and men". In: *Psychol Rev* 55.4 (July 1948), pp. 189–208.

[2]Pierre Comon. "Independent component analysis, a new concept?" In: *Signal processing* 36.3 (1994), pp. 287–314.

[3]Steven M. LaValle. "Rapidly-exploring random trees : a new tool for path planning". In: *The annual research report* (1998).

[4]Julien Mairal et al. "Supervised dictionary learning". In: *arXiv preprint arXiv:0809.3083* (2008).

[5]Andrew Ng et al. "Sparse autoencoder". In: *CS294A Lecture notes* 72.2011 (2011), pp. 1–19.

[6]Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *AISTATS*. 2011.

[7]Jakob Engel, Thomas Schöps, and Daniel Cremers. "LSD-SLAM: Large-Scale Direct Monocular SLAM". In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 834–849. ISBN: 978-3-319-10605-2.

[8]Max Jaderberg et al. "Spatial Transformer Networks". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper/2015/file/33ceb07bf4eeb3da587e268d663aba1a-Paper.pdf.

[9]Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[10]Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.

[11]Saurabh Gupta et al. "Cognitive mapping and planning for visual navigation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2616–2625.

[12]Brenden M Lake et al. "Building machines that learn and think like people". In: *Behavioral and brain sciences* 40 (2017).

[13]John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. eprint: arXiv:1707.06347.

[14]Peter Anderson et al. "On evaluation of embodied navigation agents". In: *arXiv preprint arXiv:1807.06757* (2018).

[15]Maxime Chevalier-Boisvert. *gym-miniworld environment for OpenAI Gym*. https://github.com/maximecb/gym-miniworld. 2018.

[16]Amir R. Zamir et al. "Taskonomy: Disentangling Task Transfer Learning". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2018.

[17]Manolis Savva et al. "Habitat: A Platform for Embodied AI Research". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 9338–9346. DOI: 10.1109/ICCV.2019.00943.

[18]Vidhiwar Rathour et al. "Deep Reinforcement Learning in Computer Vision: A Comprehensive Survey". In: Aug. 2021.