Multi-Object Navigation with dynamically learned neural implicit representations – Supplementary Material

Pierre Marza [1*] Laetitia Matignon[2] Olivier Simonin[1] Christian Wolf[3]
[1]INSA Lyon [2]UCBL [3]Naver Labs Europe

Project Page: https://pierremarza.github.io/projects/dynamic_implicit_representations/

1. Training stability and curves

Figure 1 shows the training curves of the 3 different agents we compared in Table 1 of the main paper: the recurrent baseline agent (blue), with the *Semantic Finder* (orange) and with both implicit representations (green). Left, (a), we see the evolution of the training reward as a mean and standard deviation over 3 runs. Right, (b) shows PPL, the main metric chosen for ranking the agents in the *MultiON* learderboard, which we show for different checkpoints during training and evaluated on the **validation set**. As can be seen, training is quite stable over runs, and adding the two representations provides a boost in performance (as already reported in Table 1 in the main paper).

2. Training procedure of the global reader

The proposed training procedure for the global reader r can be split into 3 phases. The architecture for the convolutional decoder $Dec()$ is kept the same in all of them. Table 1 (Dec) details the hyperparameters of its different layers. It is composed of 6 transpose convolution layers along with batch norm layers and ReLU activations, except for the last layer with a softmax activation. Figure 2 provides an overview of the 3 steps involved in the training of the global reader.

Fully convolutional autoencoder training on absolute maps — The first step is to train a fully convolutional autoencoder on the set of absolute maps $\mathbf{M}_i,\ i..1..25k$. Only the decoder weights are kept.

Global reader training on absolute maps The second step consists in training the global reader to output embeddings fed to the frozen decoder from the previous step. The objective is to reconstruct absolute maps from the weights of the implicit representation. The global reader weights are kept after this training phase.

Global reader finetuning on egocentric maps The global reader, whose weights are initialized from the weights obtained in the previous step, is now trained along with the same decoder from the first step (also used in the second step) on the set of egocentric maps $\mathbf{M}'_i,\ i..1..25k$. Both global reader and decoder are finetuned. The output of the global reader is not directly fed to the decoder, but is passed through linear layers in order to fuse information about first the position of the agent, and then its heading because this time the right operations of shift and rotation must be applied in order to reconstruct egocentric maps.

Integrating the pre-trained global reader into the agent After this pre-training in 3 steps, the weights of the global reader are frozen and not updated during the RL training. However, a linear layer is learnt to project the 576-dim embedding from the global reader into a 256-dim representation fed to the GRU. This linear layer is trained from reward signals.

3. Visualization of the agent behavior

3.1. Successful episode

Figure 3 provides an example for an episode rollout from the minival set of the *MultiON* CVPR 2021 Challenge. The agent is equipped with both implicit representations. For each line, from left to right, we first see the RGB-D egocentric view of the agent, then a topdown map with the three goals (white, pink and yellow squares) and their estimated location by the *Semantic Finder* (white, pink and yellow dots, with a shaded region to denote uncertainty). The radius illustrating uncertainty is unit-less and only given for visualization purposes - is not available to the agent in this particular form. The third illustration shows the implicit map obtained by querying the *Occupancy and Exploration Implicit Representation*, and on the right, there is the reconstructed output when feeding the embedding of the global

*Correspondence: pierre.marza@insa-lyon.fr

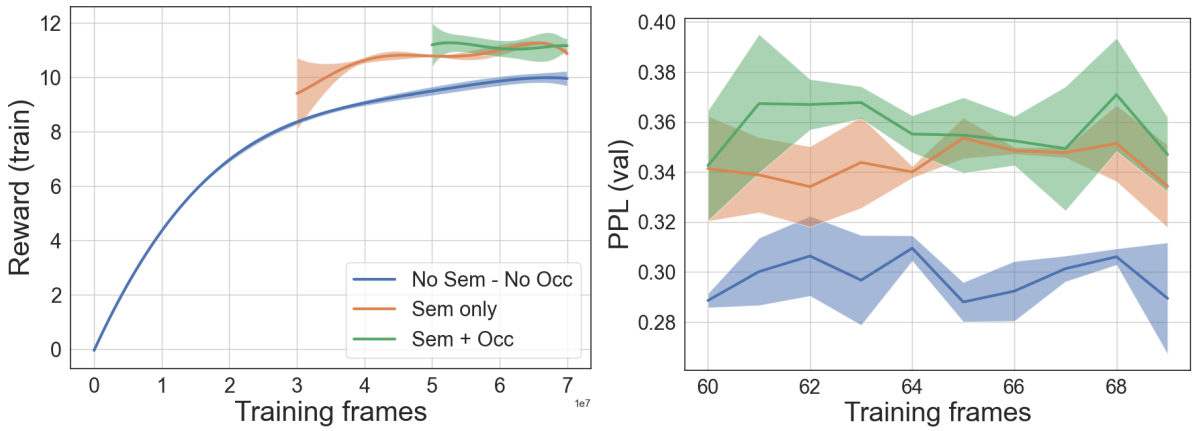

Figure 1. **Training stability:** (a) Evolution of the collected reward on training episodes for the 3 models presented in Table 1 in the main paper. (b) Evolution of PPL, the official ranking metrics in the *MultiON* Challenge leaderboard, on val episodes for model checkpoints from the last 10M training frames.

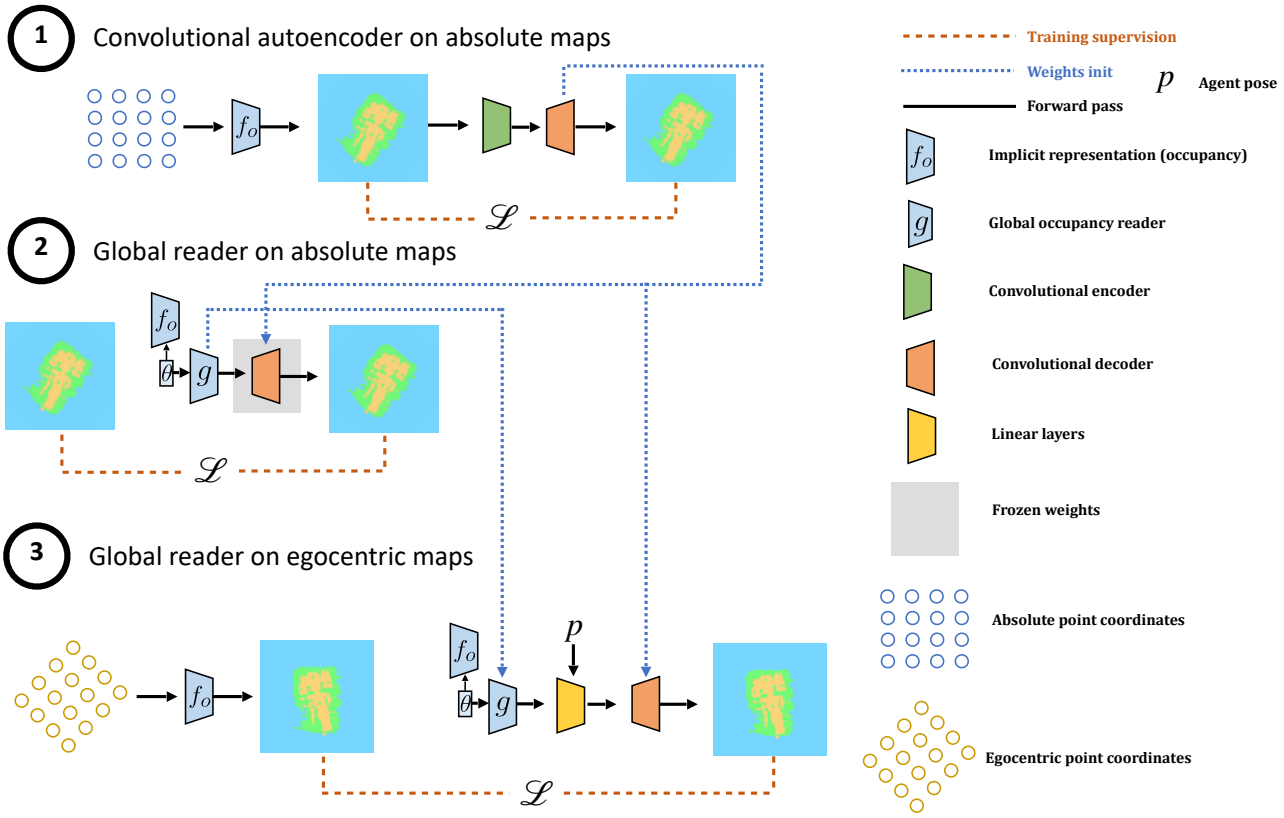

Figure 2. **Global reader training:** Overview of the 3 steps involved in the training of the global map reader.

reader to the convolutional decoder it was trained with. The last element is a curve showing the evolution of the uncertainty estimation of the *Semantic Finder* on the currently provided target object.

In this episode, the agent starts with the white object within its field of view, but the first target to reach is the pink cylinder (Row 1). As we can see, the estimation of goal positions from the *Semantic Finder* are wrong, which is expected as the episode has not yet started. However, the associated uncertainty is high, allowing the agent to discard

this information. The agent then explores until it observes the pink object (Row 2). At that point, the uncertainty about the object to find drops. The estimate of the position of the pink object will be updated as training samples will be added to the semantic replay buffer. Also note that at that point the estimate of the position of the white object from the *Semantic Finder* is accurate as the object has already been seen previously. The agent then goes towards the pink target object and calls the *found* action (Row 3). Estimation of the positions of pink and white objects are accurate. The next target to find is the white object. The uncertainty about the current target is low as the white object has already been observed. The agent backtracks (Row 4) and goes towards the white object to call the *found* action (Row 5). The next goal is the yellow cylinder. At that point, the uncertainty about the current target increases as the yellow cylinder has not yet been within the agent's field of view. The agent explores (Row 6) and when the target is within its field of view (Row 7) the uncertainty related to the target to find drops. The agent goes towards the yellow object and calls the *found* action (Row 8). At the end of the episode, the *Semantic Finder* is able to localize the 3 objects, and the associated uncertainties are low. All objects have been successfully found, so this episode is considered as a success.

3.2. Failure case

Figure 4 shows an example of unsuccessful episode. The agent and the setup are the same as described in the previous subsection. The agent can see the white target at the beginning of the episode, but it is far and largely occluded.The first target to find is the blue cylinder. It thus explores the scene until seeing the object. The uncertainty thus drops, the prediction of the target position on the map is now correct. It is interesting to see that the prediction of the location of the white target is also pretty accurate even though it was hard to detect at the beginning of the episode. The agent reaches the blue target and calls the *Found* action. The next target to find is the white object. The uncertainty is not 0 but relatively small as the agent has already seen the white target before. It goes towards the object, and when it is within its field of view, the uncertainty drops to 0 and the location prediction on the map is accurate. The agent calls the *Found* action. It has thus succeeded in finding the first two objects that were quite close to its initial position. However, as shown in the last 3 rows, the agent does not succeed in exploring the environment enough and never finds the last target which is the green cylinder. After a few steps, it calls the *Found* action at the wrong location.

4. Perception modules

Two different modules are used in this work. The first one, responsible for representation perception, extracts representations from the RGB-D observation to populate the training replay buffer of the *Semantic Finder*. The second one, tackling reactive perception, encodes the observation into a vector fed to the GRU. This representation of the observation is thus more directly used in the decision making process — the name *Reactive* is certainly not 100% accurate, since the output of this module is still used to update agent memory, but this concerns only the hidden GRU memory and not the main implicit representations.

Reactive perception We use the encoder module in [51] (ref from the main paper), which encodes visual observations at each step. Table 1 (Enc) presents the hyperparameters of the convolutional layers in this visual encoder. Is is composed of 3 convolutional layers follower by a linear layer. ReLU activations are used. The embedding produced by this visual encoder is fed to the GRU module. In our work, the reactive perception module has been pre-trained with auxiliary losses, which corresponds to the method in [34] (ref from the main paper). It is then frozen and not updated during RL training.

Representation perception The goal of the representation perception module is to extract vectors to be added to the *Semantic Finder* training replay buffer. The backbone encoder is the same as the reactive perception module (see Enc in Table 1) also pre-trained from the agent in [34] (ref from the main paper). This network is augmented with a segmentation head and is fine-tuned end-to-end on the task of segmenting *MultiON* target objects. Table 1 (Seg) details the architecture of the segmentation head. It is composed of 3 convolutional layers with ReLU activations, except for the last layer where a softmax activation is applied. After this training phase, the weights of the representation perception module are frozen and not updated during RL training.

5. Algorithmic description of an agent forward pass

Algorithm 1 gives a high-level overview of the different steps happening after receiving the current observation from the environment to take the most suitable action.

5.1. Lines $1-5$ — Adding training samples to the semantic replay buffer

The segmentation map m_t is obtained by passing the RGB-D observation o_t through the representation perception module, i.e. a segmentation model pre-trained to segment the target objects (Line 2). An inverse projection operation, denoted $invProj()$ is used to project pixels from o_t into their 3D coordinates n_t using the depth channel of o_t and the known camera intrinsics K (Line 3). A meanpooling operation,

Model	layer id	type	in channels	out channels	kernel size	stride	in padding	out padding
Dec	0	TransposeConv2D	64	32	3	2	0	0
	1	TransposeConv2D	32	32	3	2	0	0
	2	TransposeConv2D	32	16	3	2	0	0
	3	TransposeConv2D	16	8	3	2	0	0
	4	TransposeConv2D	8	8	3	2	0	0
	5	TransposeConv2D	8	3	3	2	0	1
Enc	0	Conv2D	4	32	8	4	0	–
	1	Conv2D	32	64	4	2	0	–
	2	Conv2D	64	32	3	1	0	–
Seg	0	Conv2D	32	32	5	1	2	–
	1	Conv2D	32	32	5	1	2	–
	2	Conv2D	32	9	3	1	1	–

Table 1. **Convolutional layers:** hyperparameter values in the different presented CNN architectures. Dec is the CNN decoder trained with the global reader, Enc is the visual encoder used in both the representation and reactive perceptions, Seg is the segmentation head combined with Enc in the representation perception module c.

Algorithm 1: Different steps necessary to update implicit representations at each agent step.

Input : Observation o_t, camera intrinsics K, goal g_t, replay buffers r_s and r_o, weights $\theta_{s,t}$, $\theta_{o,t}$

1 // Adding training samples to the semantic replay buffer
2 $m_t = p(o_t)$
3 $n_t = \texttt{invProj}(o_t, K)$
4 $k_t = \texttt{meanPooling}(n_t)$
5 $r_s = \texttt{addSemSamples}(r_s, m_t, k_t)$
6 // Adding training samples to the occupancy replay buffer
7 $l_t = \texttt{labelOccPos}(n_t)$
8 $r_o = \texttt{addOccSamples}(r_o, n_t, l_t)$
9 // Updating the Semantic Finder
10 **for** $i \leftarrow 0$ **to** $n_s - 1$ **do**
11 $\quad b_s = \texttt{getSemBatch}(r_s)$
12 $\quad \theta_{s,t} = \texttt{SGD}(b_s, \theta_{s,t})$
13 **end for**
14 // Updating the Occupancy and Exploration Implicit Representation
15 $loss = 0$
16 $j = 0$
17 **while** $loss > thresh$ *and* $j < n_o$ **do**
18 $\quad b_o = \texttt{getOccBatch}(r_o)$
19 $\quad loss = \texttt{eval}(b_o, \theta_{o,t})$
20 $\quad \theta_{o,t} = \texttt{SGD}(b_o, \theta_{o,t})$
21 $\quad j = j + 1$
22 **end while**

5.2. Lines $6-8$ — Adding training samples to the occupancy replay buffer

The 3D coordinates of projected pixels in n_t are compared with threshold values along their vertical y coordinate to be either labelled as navigable space or obstacle. Only 3D points with their vertical coordinate below than another threshold value are kept. These comparisons are done in the $labelOccPos()$ function (Line 7). Pairs of label and 3D coordinates are then sampled in order to keep the balance between the two classes and added to the training replay buffer of the *Occupancy and Exploration Implicit Representation*. This is the $addOccSamples()$ function (Line 8).

5.3. Lines $9-13$ — Updating the Semantic Finder

Two operations are repeated n_s times. First a batch of training examples b_s is sampled ($getSemBatch()$, line 11). Then, the $SGD()$ function encapsulates the forward pass of f_s on the sampled batch, the L1 loss computation, gradient computation and finally backpropagation. In this work, we fixed $n_s = 1$.

5.4. Lines $14-22$ — Updating the Occupancy and Exploration Implicit Representation

The implicit representation is iteratively updated for a maximum of n_o steps while the error of the model ($loss$, initialized to 0 in line 15) is higher than a threshold. Same as for the *Semantic Finder*, a training batch b_o is first sampled

denoted $meanPooling()$ is then applied to n_t in order to obtain the mean 3D coordinates of all pixels in each cell of the segmentation map m_t (Line 4). Finally, pairs of softmax distribution over classes from m_t and mean 3D coordinates are added to the training replay buffer of the *Semantic Finder*. This is implemented as the $addSemSamples()$ in the algorithm (Line 5).

($getOccBatch()$, line 18). The model is then evaluated on samples from b_o (Line 19). The $SGD()$ function is then applied to update the implicit representation. In this work, we chose $n_o = 20$ and $thresh = 0.3$.

6. Capacity of the Semantic Finder

This section provides further details about the study conducted to evaluate the impact of number of objects and the nature of their representation on the capacity of the Semantic Finder to memorize their position, and is complementary to the paragraph "*Capacity of the Semantic Finder*" in Section 4 and Figure 3 of the main paper.

To be flexible in the amount of objects we can use, we perform these experiments independently of the official *MultiON* benchmark. We consider three new scenarios, and for each one, a dataset is generated and used to train the Semantic Finder. All datasets are made of (query, position) pairs with positions being uniformly sampled between arbitrary scene bounds (between 0 and 1 along each axis). For each dataset, we also create variants varying the sample size. To reduce the amount of hyper-parameters (e.g. batch size), we ressort to gradient descent as opposed to stochastic gradient descent, i.e. each gradient step is computed over the whole dataset. The considered metrics is the mean L_1 error on the prediction of positions as a function of the number of objects to memorize. The difference between the three scenarios is in the nature of queries associated with positions, and each scenario corresponds to a sub figure of Figure 3 in the main paper.

In Figure 3a, for a given size of the dataset, i.e. for given number of objects, each query is a 1-in-K encoded vector of the object category, which means that the query dimensions grows with growing numbers of objects. This evaluates the representation in situations where objects are identified by a unique class index. Provided a sufficient number of gradient steps, we can see that the error stays low even with an increasing number of objects. We conjecture that the good performance of this setting is due to the growth in, both, query size and thus capacity of the model (as the query is the input to the model) as the number of objects grows.

In Figure 3b, the query vectors have a fixed dimension of 9, equivalent to the dimension in the *MultiON* benchmark. Queries are not 1-in-K encoded, but composed of randomly sampled values. Even though more gradient steps are helpful, the conclusion here is that increasing the number of objects has a negative impact on the mean error of the model. Unlike in (a), the number of parameters does not increase here with number of objects as queries have a fixed size. This is thus an illustration of the challenge to memorize an increasingly high number of objects with a fixed model capacity.

Figure 3c is a combination of (a) and (b) with query size increasing with number of objects and queries composed of random values (no one-hot vectors). The increase in model capacity with more objects seems again to be beneficial provided enough gradient steps. However, its is clear that the positions associated with random queries are more difficult to memorize than for one-hot queries. This emphasizes the importance of the chosen query representation when building query-able semantic implicit representations.

Optimizer	Adam
Adam eps	1e-5
Learning rate	2.5e-4
Linear learning rate decay	✓
Number of epochs	2
Number of parallel envs	16 or 4
Number of mini batches	4 or 1
Env. steps per update	128
Clipping ratio	0.2
Linear clip decay	✓
Value loss coefficient	0.5
Entropy coefficient	0.01
Max Grad Norm	0.2
GAE	✓
GAE-λ	0.95
Discount factor	0.99
Reward window size	50

Table 2. **PPO hyper-parameters:** Values for hyper-parameters used when training all agents in this work.

7. Agent training details

All agents evaluated in this work are trained with Proximal Policy Optimization (PPO) [44], following settings from previous work [51, 34]. We provide a formulation of the reward function and the PPO hyper-parameters in the next sub-sections. **Reward function** — — The reward function at time-step t is composed of three terms,

$$R_t = \mathbb{1}_t^{\text{reached}} \cdot R_{\text{goal}} + R_{\text{closer}} + R_{\text{time-penalty}} \quad (1)$$

where $\mathbb{1}_t^{\text{reached}}$ is the indicator function that equals 1 if the *Found* action was called at time t while being close enough to the target, and 0 otherwise. R_{closer} is a reward shaping term taking as value the decrease in geodesic distance to the next goal compared to previous timestep. Finally, $R_{\text{time-penalty}}$ is a negative slack reward to encourage taken paths to be as short as possible.

PPO hyper-parameters — Table 2 presents the PPO hyper-parameter values used to train all agents in this work.

8. Amount of compute

Table 3 shows the compute resources used to train, validate and test the different models involved in results presented in Table 1 and 2 in the main paper.

Type	$0-70$	$30-70$	$50-70$	S	O	Nb episodes	Nb GPUs	GPU	GPU time	Nb runs	Tot. GPU time
Train	✓			–	–		1	V100	5d	3	15d
		✓		✓	–		1	V100	4.5d	3	13.5d
			✓	✓	✓		1	V100	4d	3	12d
Val						1000	1	Titan X	4h	$90(9*10)$	15d
Test						1000	1	Titan X	4h	9	1.1d

Table 3. **Amount of compute:** GPU days for runs involved in Tables 1 and 2 in the main paper.

9. Limitations of the work

The proposed approach has the following limitations

- Slower RL training compared to baseline agents. Even if reaching an average of 45 fps with 2 implicit representations updated with backpropagation at each agent step is already quite satisfying, we are still far from the 150 fps when training $ProjNeuralMap$ or 200 fps for $NoMap$.
- The *Semantic Finder* can not deal with several instances of the same object type. This is not a problem when considering the *MultiON* task, but will be addressed in future work.
- The *Semantic Finder* only provides the position of an object of interest. Thus, it does not provide the agent with any information about how to reach the given target. Outputting a geodesic distance, and even a shortest path to the object would be interesting as future work.
- The current formulation of the uncertainty necessitates access to the full replay buffer of the episode. Future work will target estimating the uncertainty directly from the weights of the implicit representation.

10. Leveraging environment regularities and semantic priors in implicit representations

In this work, the weights of neural implicit representations are initialized from scratch at the beginning of each new episode and optimized in real time as the agent interacts with the environment. This is done in the same way as an explicit map would be updated on the fly during navigation. Training efficiency and the quality of the learnt representations are thus two important factors. Leveraging the knowledge about scene layout and semantic priors that was gained by each implicit representation to speed up the training of others and improving the quality of the provided mapping is thus a relevant future direction. Meta-learning better weight initializations, as was done in previous work (*Sitzmann et al. MetaSDF: Meta-learning Signed Distance Functions, NeurIPS 2020)*, or having a common backbone followed by randomly initialized layers for new episodes are two promising directions.

11. Importance of Fourier features

Figure 5 compares the top-down map obtained after querying the *Occupancy and Exploration Implicit Representation* trained with and without Fourier features. On each plot, the left and right maps respectively show the impact of using and not using Fourier features. Without the latter, no detail about the environment layout can be reconstructed. This corroborates findings also reported in other literature on implicit representations and coordinate networks, e.g. *(Mildenhall et al., NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, ECCV 2020)*

12. Importance of Occupancy and Exploration information

In an instance based post-hoc analysis, we attempt to visualize the way how the implicit *Occupancy and Exploration Representation* is used by the agent with two different examples.

12.1. Occupancy information

Figure 6 shows an example episode from the *MultiON* 2021 challenge minival set targeted by two different agent variants.

Without the Occupancy and Exploration Representation — the agent fails to find a target that was already observed in the past: at the beginning of the episode, the agent observes the white cylinder, while the current target is the pink object. The agent thus explores the environment, finds it and properly calls the *Found* action. The next target is then the white object. As can be seen, the *Semantic Finder* properly locates the target, since it had been previously observed. However, the agent fails to backtrack, first entering a room without finding a path to the goal and then going in the wrong direction. It finally calls the *Found* action while not being close to the white target.

The full agent — also having access to the *Occupancy and Exploration Implicit Representation* succeeds in reaching the white target (after finding the pink target, not shown on the Figure). Moreover, in visualizations of, both, the im-

plicit representation f_o and the reconstruction from the latent representation extracted by the Global Reader r, we can see the path to the goal (visualized as a red arrow) marked as explored area. All the information for the agent to correctly navigate towards the white object is thus contained in the implicit map and its global summary vector.

This analysis cannot corroborate that the agent indeed used the representation as explained; however, we can at least provide evidence that the (successful) full agent had access to information, which was crucial to solve a task, on which the baseline agent failed.

12.2. Exploration information

Figure 7 shows another example episode taken from the *MultiON* 2021 challenge minival set.

Without the Occupancy and Exploration Representation — the agent fails to explore the scene to find a target: the first target object is black. To this end, the agent must explore the environment. It starts to explore but misses a part of the scene (containing the black target), which is never explored for the rest of the episode. It finally calls the *Found* action far away from the black target.

The full agent — also having access to the *Occupancy and Exploration Implicit Representation* succeeds in finding the black target. It successfully explores the part of the scene that contains the object. In visualizations of, both, the implicit representation f_o and the reconstruction from the latent vector extracted by the Global Reader r, we can see that the area to observe (visualized by a red shape) is at the frontier between explored and unexplored parts of the scene. This information can thus guide the agent to move towards the unexplored area.

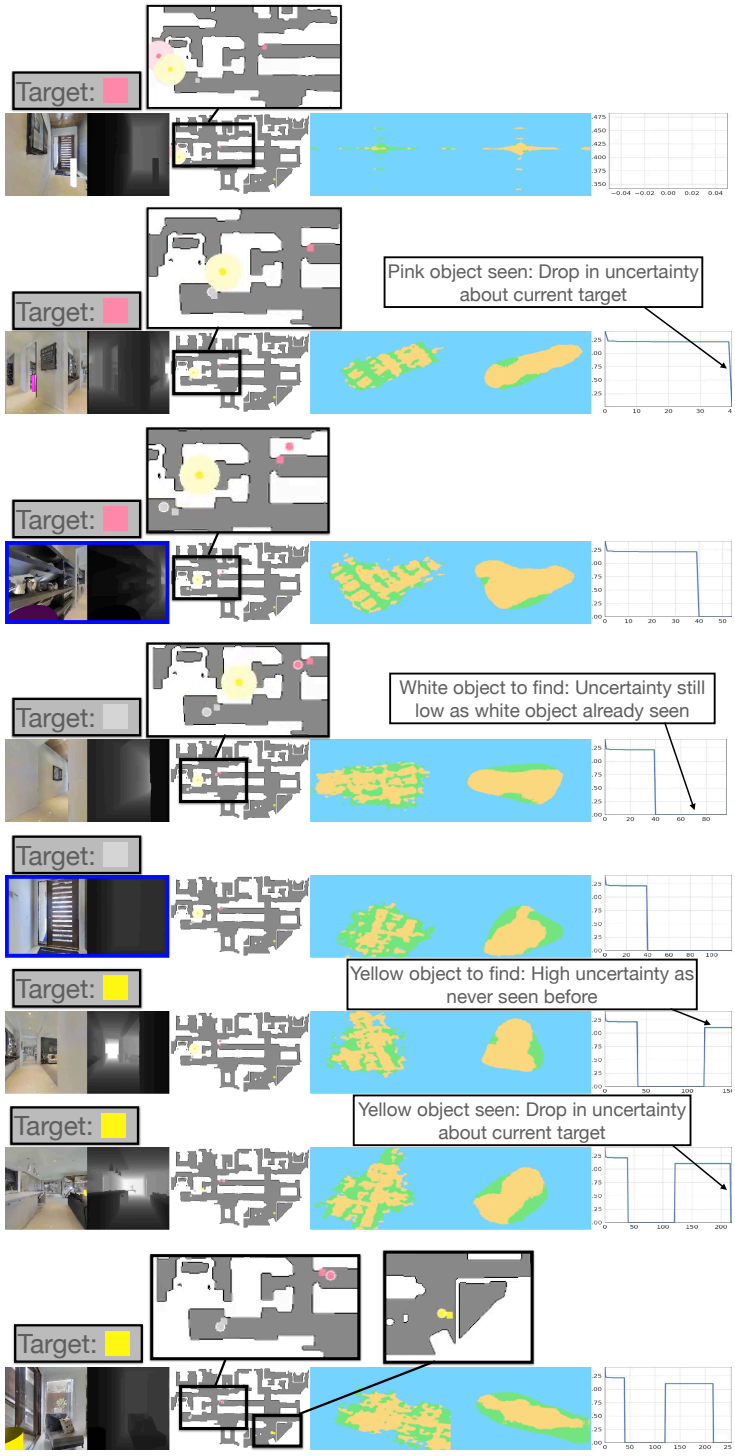

Figure 3. **Agent rollout on an example successful episode** from the *MultiON* CVPR 2021 Challenge minival set. From left to right: RGB-D ego view, topdown map (viz only) with targets (squares) and their estimated location by the *Semantic Finder* (dots, shaded region for uncertainty), map from the *Occupancy and Exploration Implicit Representation*, reconstructed egomap from the global reader and CNN Decoder trained end-to-end, uncertainty of the *Semantic Finder* on the currently selected target.

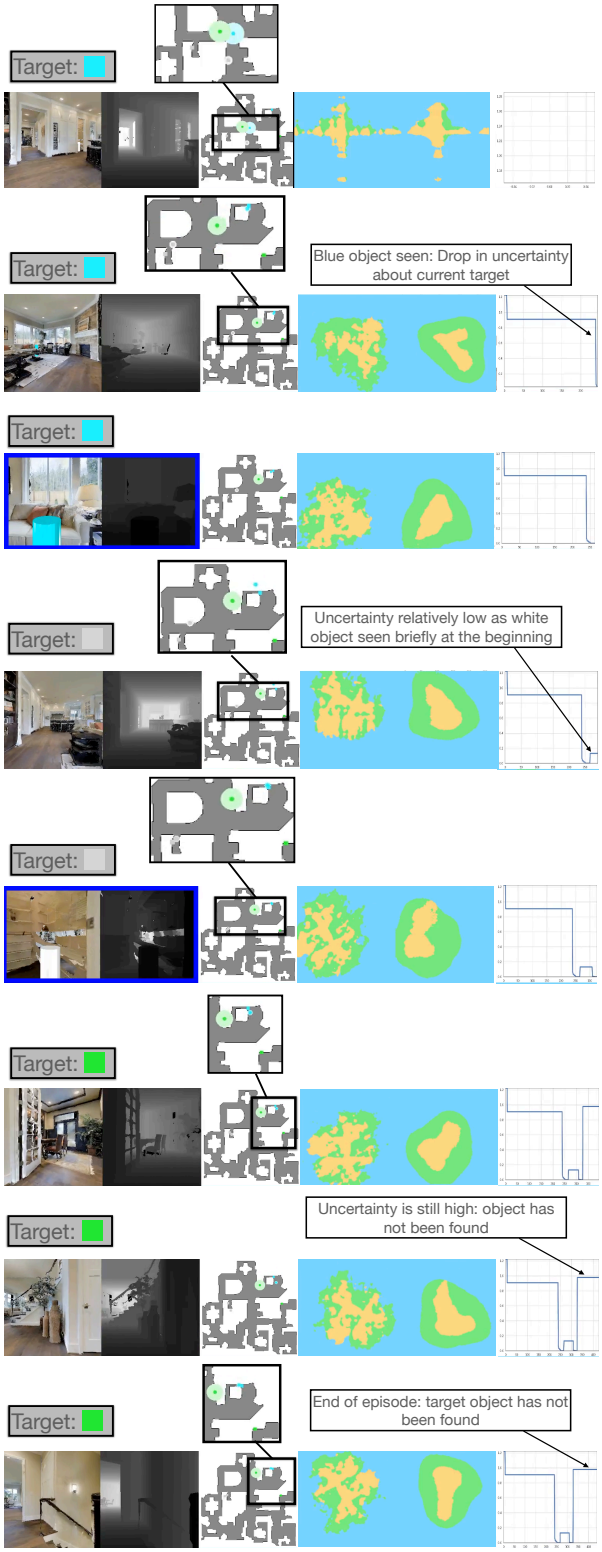

Figure 4. **Agent rollout on an example unsuccessful episode** from the *MultiON* CVPR 2021 Challenge minival set. From left to right: RGB-D ego view, topdown map (viz only) with targets (squares) and their estimated location by the *Semantic Finder* (dots, shaded region for uncertainty), map from the *Occupancy and Exploration Implicit Representation*, reconstructed egomap from the global reader and CNN Decoder trained end-to-end, uncertainty of the *Semantic Finder* on the currently selected target.

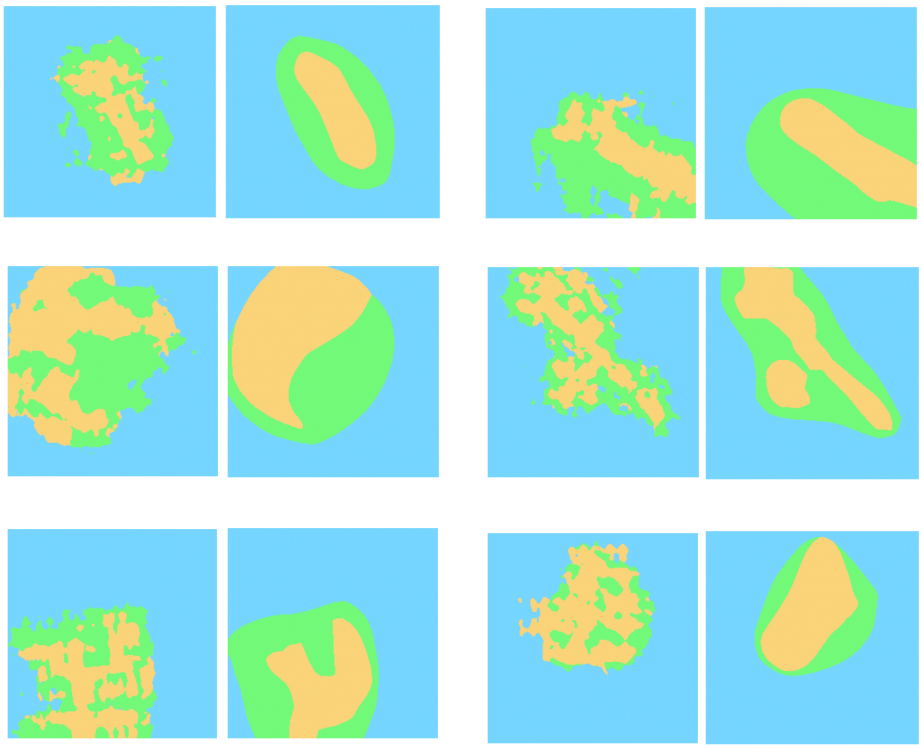

Figure 5. Comparison of top-down maps obtained by querying the *Occupancy and Exploration Implicit Representation* trained with (left) and without (right) Fourier features.

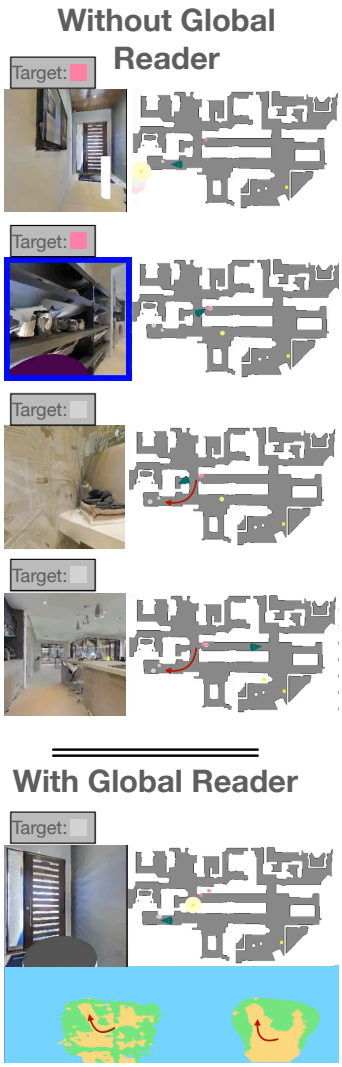

Figure 6. **Importance of occupancy information**: Episode example from the *MultiON* 2021 challenge minival set where an agent without the proposed *Occupancy and Exploration Implicit Representation* fails to find the white target. Another agent equipped with the representation containing occupancy information finds the target. Some information about the path to reach it (indicated with the red arrow) is indeed contained in the map. The green cone corresponds to the agent's position.

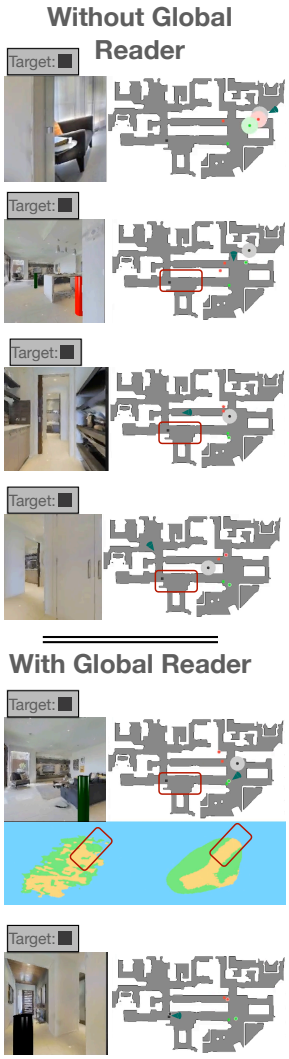

Figure 7. **Importance of exploration information**: Episode example from the *MultiON* 2021 challenge minival set where an agent without the proposed *Occupancy and Exploration Implicit Representation* fails to find the black target. Another agent equipped with the representation containing occupancy information finds the target. Some information about the area to explore (indicated with the red shape) is indeed contained in the map. The green cone corresponds to the agent's position.