Learning to Navigate with Environment Map Memory

Xiaoyang Guo
Department of Electronic Engineering
The Chinese University of Hong Kong
Shatin, Hong Kong
xyguo@link.cuhk.edu.hk

Yi Zhang
Department of Electronic Engineering
The Chinese University of Hong Kong
Shatin, Hong Kong
zhangyi@link.cuhk.edu.hk

Abstract

Visual navigation is a fundamental problem for robotic control. It tries to navigate an agent to the target in an environment using camera inputs. Previously, the navigation problem is solved by optimal route optimization with a geometric reconstruction of the world. However, the error in the reconstruction will result in sub-optimal navigation routes or even failures, and it is difficult to handle unknown environments and highly dynamic obstacles. Some recent work proposed to utilize deep reinforcement learning (RL) to teach the agent to navigate to the targets without any prior information. In this project, we propose to improve the RL navigation performance by employing environment map memory, which is a graph built by the feature representations of all visited places during the exploration stage. During the exploration stage, the agent is allowed to explore freely to exploit the environment. During the navigation stage, the memory graph is utilized to provide hints about next optimal actions for the navigation policy network. Our experiment results show that the graph representation provides useful information to improve the visual navigation performance.

1 Introduction

Visual navigation is the problem that makes the agents (*e.g.*robots) to explore the environment by itself to reach the given targets. The algorithm usually takes raw image frames as inputs. Traditional methods usually optimize optimal routes given a geometric representation of the environment. The geometric representation could be pre-defined by 3D reconstruction or built with on-the-fly SLAM maps. However, these methods are time-consuming and easily affected by noises in the maps, which will cause errors or even failures for the navigation task. In recent years, deep reinforcement learning algorithms were exploited to handle dynamic objects and obstacles to improve the navigation performance in complex environments.

However, the recent deep visual navigation algorithms are either supervised trained [4] or directly training a policy network without any map representation about the whole environment [7]. Imagine the agent receives a command to look for a trash bin, and the agent has already seen the target before. Actually this kind of auxiliary memory information could help the agent find the target more easily. In this paper, what we are trying to do is to log all of the previously visited places in a semantic graph, then the graph is utilized to provide auxiliary information for the navigation policy network to improve the performance.

There are different kinds of visual navigation problems in previous papers. For example, some papers work on known environments, while some other papers work on unknown environments or allow the agent to explore the environment in limited time. Some may provide the agent with target position coordinates, while recent deep-based algorithms mainly take raw target images or word embedding

of the targets as inputs. In our project, we take the word embedding of the target (*e.g.*refrigerator) and raw images of the environment as inputs and evaluate our algorithms in unknown environments.

In this paper, different from previous RL navigation papers, we will allow the agent to explore in the new environment freely before assigning targets to the agent. With the routes during the exploration stage, we build a environment map memory graph as a global representation of the whole environment, which is composed of feature presentations of all visited places during the exploration stage. The graph could provide some noisy information of optimal routes to the target objects, which is taken as inputs in our policy network as auxiliary inputs. Once the agent is specified with a target, the policy network will utilize recent states (raw images), target word embedding, and the memory graph to predict the next action. Experiment results show that the auxiliary inputs provided by the map memory graph is beneficial to improve the navigation performance.

2 Related Work

Recently deep reinforcement learning is applied to make agents to learn all kinds of control tasks, such as robot locomotion, obstacle detection, playing ATARI games, etc.. Compared to traditional reinforcement learning, deep reinforcement learning leverages the representation and encoding power of neural networks to boost the performance of traditional reinforcement learning algorithms. Deep Q-learning utilizes a neural network to approximate the state-values function (Q-value function). The training of deep Q-learning can be stabilized by employing a target network and experience replay. There are also many policy-based algorithms, including REINFORCE, etc. Many algorithms try to improve policy-based methods with learned state-value functions used as the baseline in the policy gradient update, including A3C, A2C.

There are roughly two main categories for the navigation task. One branch is the map-based navigation, which solves the optimal route with geometric reconstruction or maps in other formats. The maps may be pre-defined or built on the fly. Another branch of work is based on reinforcement learning. Explicit reconstruction or maps of the environments are not required. Zhu *et al*. [8] explored the problem of navigating a space to find a given target goal using only visual input. The authors created AI2-THOR environment, and proposed an actor-critic model whose policy is a function of the goal as well as the current state to improve the generalization capability. Kulhanek *et al*. [3] tried to train an agent to navigate based on the observed raw images only, which is trained by a batched version of advantage actor-critic (A2C) [6]. The features of both the environment and the visual target are extracted and matched by neural networks. Tanderson *et al*. [1] trained a model to navigate the agent with natural language inputs. Sax *et al*.proposed to utilize mid-level priors such as occlusions, normal maps and edge maps as auxiliary inputs for the navigation policy network, which makes the model generalizes better in novel scenes. Qiu *et al*.exploits object relation graph to help the agent find objects nearby the target, which improves the success rate of visual navigation.

Another branch of approaches utilize map for long-range navigation. This could be done with pre-define reconstruction of the environment and online built maps with Simultaneous Localization and Mapping (SLAM) [2] techniques. For this kind of traditional methods, path planning algorithms could be employed to find the optimal path to the final target. However, generating a geometric graph is not trivial, and sometimes small errors in the graph will result in wrong routes or even failures. Recently, there are some works on utilizing graph for deep reinforcement learning based visual navigation. Savinov *et al*. [4] introduces a semi-parametric topological memory, which consists of a non-parametric graph with nodes corresponding to locations in the environment and a deep network to retrieve the node in the graph. However, the network is trained with direct supervised labels, which may result in sub-optimal results in unseen circumstances. In this paper, we expand this paper to train the policy network with reinforcement learning.

3 Approach

3.1 Task Definition

Given an unseen environment E, we allow the agent to explore freely for at most 200 steps. After the exploration stage, a object target t is specified, for example 'refrigerator', the agent should navigate to the target with as fewer steps as possible. Once the agent reaches the target, it should perform the **DONE** action to notify the environment that it has already found the target, otherwise, it cannot be

counted as success because the environment does not know the target object is visible in the current frame. If there are multiple available targets in the environment, finding any one can be counted as success.

3.2 The baseline without exploration

We pick the baseline from [4] as our baseline model. The structure of the baseline model is shown in Figure 3a. There are two branches of inputs. The first branch takes a sequence of images and extract features using ResNet-18. The second branch encodes the target word using Glove embedding, and then transform it with a fully connected layer. The features from two branches are then concatenated to form a sequence of fused features. In order to take sequential information into consideration, a LSTM model is applied to process the feature sequence. Finally, the action is predicted from transformed sequential features.

The predicted actions include turn left/right, look up/down, go forward, and **DONE** action. The **DONE** action is used by the agent to confirm the navigation process is done, then the environment will check whether the target object is visible in the current view and gives reward accordingly. The environment will give a large reward of 5 if the agent successfully find the target, and a small penalty -0.01 in other cases. This policy network is trained with A3C algorithm to improve parallel sample efficiency.

3.3 Proposed Method

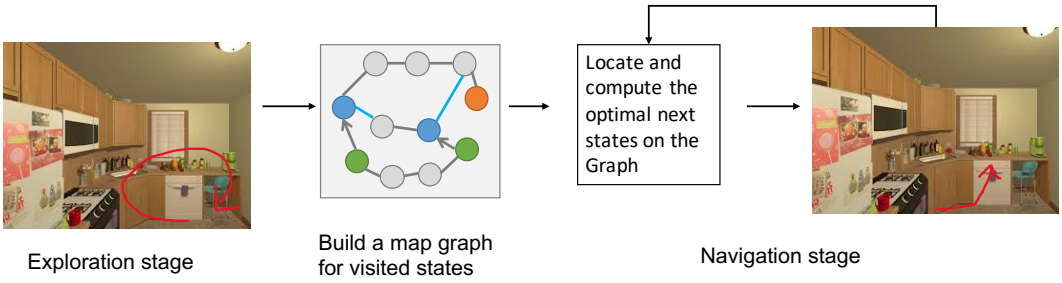


Figure 1: Illustration of the proposed algorithm. The whole algorithm is mainly divided into 3 steps, exploration stage, build the map memory graph, and navigation stage. In the navigation stage, the algorithm will iteratively compute the potential optimal next steps for the current state using the graph information and feed them into the policy network as auxiliary feature inputs.

Different from previous RL-based visual navigation approaches [4, 7], we add an extra exploration stage to make the agent explore the unseen environment freely, which is introducted in Sec. 3.3.1. The visited states are utilized to build a map memory graph in Sec. 3.3.2. After these two preparation stages, the agent is specified with targets and the navigation stage starts. The navigation stage will iteratively compute the potential optimal next steps for the current state using the map graph, which is fed into the policy network as auxiliary feature inputs to improve the performance. Details of navigation stage are described in Sec. 3.4. The whole algorithm is described in Alg. 0 in details.

3.3.1 Exploration Stage

The target of the exploration stage is to make the agent visit as many new parts of the environment as quickly as possible. In previous papers, exploration agent training usually partition the whole space into small occupancy cells, and the environment will decide whether a cell is visible to the agent in the current state or not. According to the number of new explored visible cells, the environment will return corresponding rewards to the agent. Different from previous methods, since our final objective is to find the target objects, we hope the agent to find as many candidate target objects as possible instead of occupancy cells. As a result, the exploration reward is defined based on whether the agent finds unseen objects.

Assume the total number of candidate objects in the current environment E is N_{obj}, then the reward for each step is defined as follows,

$$R_{\text{explore}} = \begin{cases} 10/N_{obj} & \text{if an unseen object is visible,} \\ -0.01 & \text{otherwise.} \end{cases} \quad (1)$$

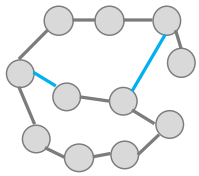
For each step, if there is one new object is visible, then there will be a positive reward $10/N$, otherwise, there will be a small penalty reward -0.01. The network structure in the exploration stage is very similar to the baseline model in Fig. 3a, except that the target embedding branch is removed since there is no specific target during the exploration stage.

3.3.2 Build map graph

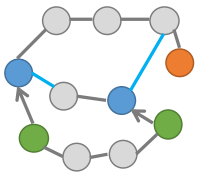
We employ map memory graph as representations of the current environment. Assume we have an exploration sequence $s_0, s_1, ..., s_n$, then each state is represented as a node in the graph, as shown in Fig. 2a. There exist an edge between s_i and s_j only when s_i and s_j are adjacent states in the sequence or the similarity score between these two states are very high. As shown in Fig. 2a, the gray edges are built from adjacent states, and the blue edges are added for two states with highest similarity scores. The reason why we add blue shortcut edges is that sometimes the agent visits the same place twice during exploration. The feature representations of each visited state s_i is also recorded in the graph, which is denoted by F_{s_i}.

In order to measure the similarity scores between two states, we train a small network S_f in a self-supervised way. The network takes features of two images as inputs and outputs a similarity score between 0 and 1. Firstly we randomly sample several sequences, then we make the network S_f to predict whether the distance between two input features is smaller than 2 or not. By doing this, we can obtain a neural network which estimates the similarity score between two states.

We also train another network S_o, which describes the similarities between image features and word embedding of target objects. This model is used to find the potential target state with the target word embedding in the graph, as the orange node in Fig. 2b.



(a) An illustration of map memory graph. Each node is a state in the exploration sequence. The gray edges are built from adajacent states, and the blue edges are added for two states with highest similarity scores.



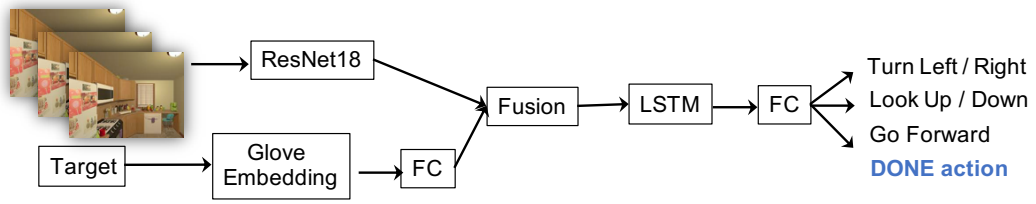
(b) To provide optimal next-step candidates, we need to find the target state using S_o and the current state using S_f in the graph. The orange node represents the potential target node. The green nodes denote potential current states.

Figure 2: The map memory graph.

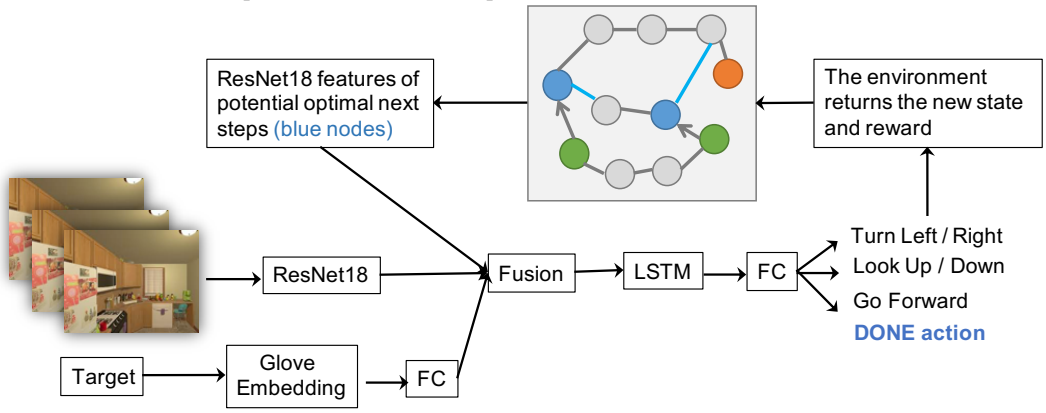
3.4 Navigation with Map Memory Graph

To make the agent better explore in new environments, we will allow the agent to explore the environment in limited steps. From Sec. 3.3.1 and 3.3.2, we have made the agent explore the environment with exploration policy and saved useful features into a data structure called map memory graph. In the navigation stage, the map memory graph is fixed. The agent is provided with a target and utilizes the map memory graph to predict the next action.

As shown in Alg. 0 and Fig. 3b, the navigation stage will iteratively compute the potential optimal next steps for the current state using the map memory graph, which is then fed into the policy network as auxiliary feature inputs to improve the performance. Next we describe the details step by step. In each iteration of the loop, first we need to compute the similarity scores between the target embedding and each of the nodes in the graph to find the potential target node, which is the orange node in Fig. 2b. Then we need to locate the current state in the graph by comparing the current image feature representations with each of the nodes in the graph. Since the relocalization may bring a lot of noises, we pick 3 candidates c_0, c_1, c_2 with top-3 similarity scores in the graph, shown as green nodes in Fig. 2b (only 2 shown in the figure). For each candidate c_i, we solve the optimal route from c_i to the



(a) The baseline method. The baseline model simply takes a sequence of images as inputs, then extracts the features of images using ResNet18. The target word is converted to feature vector using Glove embedding. The two features are fused and processed with LSTM to predict actions.



(b) The structure of the policy network in navigation stage.

target t with Dijkstra's algorithm, which is denoted by $r_i^0, r_i^1, ..., r_i^k$. Note that the first state r_i^0 in the optimal route is the starting node c_i itself. We pick the optimal next step r_i^1 as hints for the policy network, as shown as the blue nodes in Fig. 2b and the first input branch of network in Fig. 3b. From Fig. 3b, we can see the only difference between the baseline and our method is the extra input branch of image feature representations of potential optimal next steps.

Algorithm 0: Visual navigation with map memory graph

Initialize the environment;
Save the map graph G, and connect shortcuts based on image feature similarity model S_f;
Find the target node t in the graph G based on object-to-image similarity model S_o;
while a *is not* ***DONE*** **do**
    Find the current potential states c_0, c_1, c_2 in the graph G based on S_f;
    foreach c_i *in* c_0, c_1, c_2 **do**
        Solve the optimal route from c_i to t with Dijkstra's algorithm: $r_i^0, r_i^1, ..., r_i^k$;
        Pick the optimal next step r_i^1 of c_i;
    end
    Input the image feature representations of the current frame F_c, potenital optimal next steps $F_{r_0^1}, F_{r_1^1}, F_{r_2^1}$, and the target object embedding E_t into the policy network P to predict the next action a;
    The environment returns the reward r.
end

4 Experiments

4.1 Implementation Details

Following [5], we employ A3C to implement our algorithm due to its high parallel efficiency. Both of the exploration policy network and the navigation policy network is trained with Adam optimizer

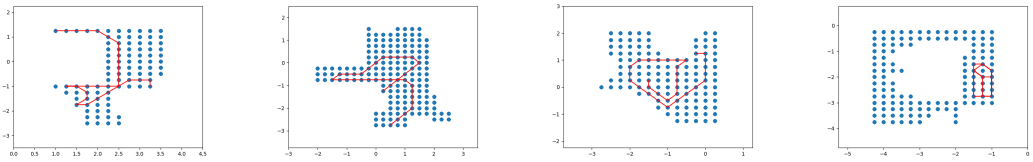
for 160 epochs. The learning rate is set to 1e-4. The network structure follows [5] and the number of convolution filters is not modified.

4.2 Environment and Data

We use a discrete version of AI2-THOR [8] as our experiment environment, following [4]. It provides 4 types of indoor scenes including kitchens, bedrooms, bathrooms, and living rooms. Over 2000 unique objects are available for AI agents to interact with. For each room type, there are 30 shapes in total. We split the rooms with id 1-20 for training, 21-25 for validation, and 26-30 for testing. In Fig. 4, we show some examples of room shapes in bird view, in which blue dots are available discrete states and red lines are exploration sequences by our exploration agent. For each blue dot, there may exist multiple states with different directions and viewpoints.

4.3 Exploration Analysis

Here we show several samples of exploration sequences in Fig. 4. For the first 3 examples (a), (b) and (c) we can see the agent successfully find most of the candidate objects by visiting each corner of the room. However, there are also several failure cases, as shown in (d), the exploration agent sometimes gets stuck in the right part of the environment and fails to give a good exploration sequence.



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

Figure 4: Examples of discrete environment (bird views) and exploration sequence. The blue dots denote available discrete states (may contains multiple states with different viewpoints). The red line shows the exploration route. In (a)(b)(c), the exploration sequence successfully finds most of the candidate objects. While (d), which is a failure case, gets stuck in the right part of the environment and fails to give a good exploration sequence.

4.4 Evaluation Metrics

There are two evaluation metrics, including success rate and SPL (success rate weighted by path length). Success rate is the ratio of success trials. A trial is counted as success only when the agent find the target and carries out the DONE action at the same time. SPL is the success rate weighted by path length, which is defined as follows,

$$\text{SPL} = \frac{1}{N} \sum S_i \frac{L_i}{\max(L_i, P_i)}, \tag{2}$$

in which S_i is the success indicator (1 for success and 0 for failure) of the ith trial, P_i is the navigation path length, and L_i is the optimal path length. There are N trials in total.

4.5 Results & Analysis

In Table 1, we show the final performance of our model and compare with previous methods. The baseline is described in Sec. 3.2. Scene Priors [7] explore the relationship between objects to provide prior information about the target object. SAVN [5] proposed to utilize meta-learning to perform self-adaptation to solve the generalization problem in the visual navigation task. *All* denotes all the trials and $L \geq 5$ only considers the trials whose optimal path length is greater than 5.

From the results we can see that our proposed method SPL by 0.99% and success rate by 4%, which show the effectiveness of the proposed method. Similar conclusion can be drawn from the results of $L \geq 5$. However, the performance of our method is still far from the state-of-the-arts.

Table 1: Performance comparison between different methods. Note that SAVN utilizes meta-learning to do online domain adaptation. The baseline model is from [5], as described in Sec. 3.2

	All		L≥5	
	SPL (%)	Success (%)	SPL (%)	Success (%)
The baseline [5]	14.68	33.04	11.69	21.44
Scene Priors [7]	15.47	35.13	11.37	22.26
SAVN [5]	16.15	40.86	13.91	28.70
Ours	15.67	37.03	13.10	24.23

5 Conclusion

In this project, we built a deep visual navigation framework which utilizes environment map memory graph to provide a global semantic representation of the whole environment, which provides useful hints for the navigation agent. Experiments show that our proposed method improves the performance of the visual navigation task. However, there are several problems in our implementation, the accuracy of relocalization of the current state is not high enough, which may be solved by some robust semantic SLAM algorithms in the future.

References

[1] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3683, 2018.

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

[3] J. Kulhánek, E. Derner, T. de Bruin, and R. Babuška. Vision-based navigation using deep reinforcement learning. In *2019 European Conference on Mobile Robots (ECMR)*, pages 1–8. IEEE, 2019.

[4] N. Savinov, A. Dosovitskiy, and V. Koltun. Semi-parametric topological memory for navigation. *arXiv preprint arXiv:1803.00653*, 2018.

[5] M. Wortsman, K. Ehsani, M. Rastegari, A. Farhadi, and R. Mottaghi. Learning to learn how to learn: Self-adaptive visual navigation using meta-learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[6] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pages 5279–5288, 2017.

[7] W. Yang, X. Wang, A. Farhadi, A. Gupta, and R. Mottaghi. Visual semantic navigation using scene priors. *arXiv preprint arXiv:1810.06543*, 2018.

[8] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.