Modular Visual Navigation using Active Neural Mapping

Devendra Singh Chaplot\textsuperscript{1,2}, Saurabh Gupta\textsuperscript{2,3}, Abhinav Gupta\textsuperscript{1,2}, Ruslan Salakhutdinov\textsuperscript{1}\textsuperscript{*}
\textsuperscript{1}Carnegie Mellon University, \textsuperscript{2}Facebook AI Research, \textsuperscript{3}UIUC

Abstract

Learning-based navigation algorithms exploit the semantic structure of the world and are effective at handling sensor noise and noisy maps. However, they are sample inefficient, do not transfer across domains, and fail at long-term planning. On the other hand, classical algorithms are effective at long-term planning and need little to no training data but do not exploit the semantic structure and fail with noisy maps/observations. In this work, we present a modular and hierarchical navigation algorithm which leverages the strengths of both classical and learning-based methods. Our model consists of long-term+short-term goal generators and long-term+short-term planners. We use learning to generate long-term goals and a deterministic planner to solve for long-term paths. Short-term goals are generated based on the planned paths but the short-term planner is learned, making it robust to sensor noise. Our modular approach provides efficient, exhaustive exploration; accurate long-term planning; and transferability across domains and tasks. We perform experiments on the Gibson and Matterport real-world reconstruction datasets in the Habitat simulator. We show that the proposed model outperforms prior methods on both exploration (43\% relative improvement in coverage) and PointGoal navigation (21\% absolute improvement in success rate), while also improving sample efficiency\textsuperscript{2}.

1 Introduction

Navigation is a critical task in building intelligent agents. Agents need navigation to move to new locations in known/unknown environments and also to explore new environments. But what are the desirable properties of a navigation algorithm? First, we would like a navigation model to be effective at both point-goal task and exploration, i.e. capable of both searching the environment efficiently when the goal location is not known and navigating effectively to the goal when its location is known. Second, a navigation model should exhibit strong generalization performance. Generalization can be of different types such as generalization to new scenes, domains, and tasks. Finally, we would like the model to be sample efficient as to minimize the amount of data required for both training the model and transferring it to new domains or tasks.

Classically, navigation methods are mostly based on deterministic planning algorithms such as Dijkstra\textsuperscript{[9]}, A-Star\textsuperscript{[15]} and exploration algorithms such as Frontier-based Exploration\textsuperscript{[46]}. These approaches however are brittle to sensor noise and do not exploit any semantic structure in the world (hallways lead to rooms, kitchens near dining area, etc.). Therefore, in recent years there has been a focus on learning-based approaches for navigation. End-to-end learning models are shown to be effective at the PointGoal task trained using imitation learning\textsuperscript{[14]} and reinforcement learning\textsuperscript{[38]}. Learning-based methods are especially very effective as compared to classical methods in the unknown goal position tasks as these tasks involve additional challenges such as object

\textsuperscript{*}Correspondence: chaplot@cs.cmu.edu
\textsuperscript{2}See \url{https://sites.google.com/view/active-neural-mapping} for visualization videos

Preprint. Under review.
recognition \[26, 29\], grounding \[19, 5\], and reasoning \[8, 13\], where it is difficult to hand-design a navigation policy using classical approaches. However, learning-based approaches have certain limitations. First, they are highly sample inefficient and typically require hundreds of millions of frames for training. Second, even if we manage to train such a policy, these policies are extremely task-specific. They require another tens of millions of samples to be finetuned or adapted to a new task, and exhibit catastrophic forgetting \[35, 32\]. Finally, learning-based approaches are ineffective at long-term planning and suffer because of exponential explosion in search space as the distance to goal increases.

Interestingly, the shortcomings of learning-based approaches are the strengths of classical approaches. Specifically, classical approaches are effective at exhaustive exploration and long-term planning, require very little or no training data and can be used for different tasks. Inspired by these observations, in this paper, we design a modular hierarchical navigation model that leverages the strengths of both learning-based and classical methods. Our hierarchical model consists of both long-term+short-term goal generators and long-term+short-term planners. Long-term goals are either given or generated by a learned-module which exploits semantic structure and the long-term planning is achieved via deterministic planning. Short-term goals are generated based on the planner’s path but the short-term policy (local policy) is a learned policy which is robust to sensor noise. This modular structure allows us to leverage the strengths of classical planning algorithms to overcome the limitation of end-to-end learning methods. Our modular approach has the following advantages:

**Efficient and exhaustive exploration:** Our global goal-generator samples long-term goals. It uses learning and hence exploits the semantic structure which makes exploration exhaustive and efficient. Generating long-term goals reduces the time horizon for exploration as compared to predicting low-level navigation actions, making it sample-efficient.

**Accurate long-term planning:** Planner acts as a natural link between the global and short-term goals. Use of deterministic planning ensures high-quality paths and robustness to increasing map-size.

**Domain and Task Generalization:** The local policy is only trained to reach the short-term goal. This makes the local policy task-invariant; the onus of task-dependency is on the global policy that can select different long-term goals based on the task, while the local policy can be transferred across tasks without any fine-tuning. On the other hand, the global goal-generator is domain-invariant as it only looks at the spatial obstacle map predicted by the Mapper and thus, can be transferred across different domains. The modularity of the model allows us to transfer the knowledge of obstacle avoidance and control in low-level navigation across tasks and the knowledge of high-level exploration policies across domains.

The above advantages enable the proposed model to outperform prior methods on both Exploration (43% relative improvement in coverage) and PointGoal navigation (21% absolute improvement in success rate) on the Gibson dataset, while also improving sample efficiency. As compared to prior methods trained separately for each task, the proposed method needs to be trained only for Exploration and can be directly transferred to the PointGoal task without any additional training. We also show that the proposed model outperforms all the baselines at generalization to the Matterport domain and generalization to harder goals.

## 2 Related Work

Navigation has been well studied in classical robotics. There has been a renewed interest in the use of learning to arrive at navigation policies, for a variety of tasks. Our work builds upon concepts in classical robotics and learning for navigation. We survey related works below.

**Classical Approaches.** Classical approaches to navigation decompose the problem into two parts: map building and path planning. Map building is done via simultaneous localization and mapping \[43, 16, 12\], by fusing information from multiple views of the environment. Mapping typically requires specialized scanners such as LiDARs or Kinect \[20\], which are used to generate detailed occupancy maps. Such maps can be used for path planning \[22, 27, 3\] to compute paths to known goal locations. Classical systems often require manual task-specific tuning and are brittle to noise in the depth sensor as it leads to inaccuracies in the constructed spatial occupancy maps. Researchers have also pursued topological maps \[24\] and spatio-semantic maps \[25\].
We propose a modular navigation model, ‘Active Neural Mapping’. It consists of four components: a Mapper which updates the current map based on the current observation, a Global policy which uses the predicted map to produce a long-term goal, a Planner computes a plan to reach the current long-term goal and produces a short-term goal on this path, and a Local policy which takes navigational action based on the current observation to reach the short-term goal. See Figure 1 for an overview.

Classical methods have inspired a number of learning based techniques. Researchers have designed neural network navigation policies that reason via spatial representations [14, 31, 47, 18, 13], and topological representations [36, 37], differentiable and trainable planners [42, 28, 14, 23]. Our work furthers this investigation, and we investigate a hierarchical decomposition of the problem that combines learning with classical deterministic planning algorithms such as the Fast Marching Method [41]. As these planners are deterministic and provably accurate, the learning burden is significantly reduced. This allows us to obtain better performance at navigation tasks and achieve strong generalization to new domains and goals, while improving the sample efficiency.

### Navigation Tasks

Given the renewed interest in using learning for navigation related tasks, a number of problem definitions and setups have emerged. These can broadly be classified into two categories: known goal location, and unknown goal location. [14, 38, 2, 69, 30, 33] investigate the known goal location task, where goal has been specified explicitly as a point in space, or implicitly as a sequence of images, or by language instructions specifying the path to the goal. These tasks do not require exhaustive exploration. On the other hand, many works consider the latter task with unknown goal location, where goal is to navigate to a fixed set of objects [26, 10, 44, 29, 14], navigate to an object specified by language [19, 5] or by an image [48], or to navigate to answer a question [8, 13], or to explicitly explore the environment [6, 11, 36]. Tasks in this category essentially involve effective exploration of the environment. While end-to-end reinforcement learning is shown to be effective at these tasks when the goals are spawned close to the agent, exhaustive exploration and long-term planning in large environments with distant goals is challenging. Furthermore, most of these learning based policies are specialized to the task they are trained for. In contrast, our navigation model is capable of handling both the pointgoal and exploration tasks.

## 3 Methods

A navigation model takes in an observation $s_t$ at each time step $t$ and outputs a navigational action $a_t$. Observations $s_t$ are typically RGB images showing the first-person view of the environment. We would like our model to handle multiple navigational tasks $T_i \in T$. The objective is to learn a policy $\pi(a_t | s_t, T_i)$ which is optimal at performing the task $T_i$.

We propose a modular navigation model, ‘Active Neural Mapping’. It consists of four components: a Mapper which updates the current map based on the current observation, a Global policy which uses the predicted map to produce a long-term goal, a Planner computes a path to reach the current long-term goal and produces a short-term goal on this path, and a Local policy which takes navigational action based on the current observation to reach the short-term goal. See Figure 1 for an overview.

The Active Neural Mapping model internally maintains a spatial map, $m_t$ and pose of the agent $x_t$. The spatial map, $m_t$, is a $2 \times M \times M$ matrix where $M \times M$ denotes the map size and each element in this spatial map corresponds to a cell of size $25cm^2$ in the physical world. Each element in the first channel denotes the probability of an obstacle at the corresponding location and each element in the second channel denotes the probability of that location being explored. A cell is considered to be explored when it is known to be free space or an obstacle. The spatial map is initialized with all zeros at the beginning of an episode, $m_0 = [0]^{2 \times M \times M}$. The pose $x_t \in \mathbb{R}^3$ denotes the $x$ and $y$ coordinates of the agent and the orientation of the agent at time $t$. The agent always starts at the center of the map facing east at the beginning of the episode, $x_0 = (M/2, M/2, 0.0)$. The pose of the agent is transformed based on the previous action using a transition function, $f_t$ i.e. $x_{t+1} = f_t(x_t, a_t)$.
Mapper. The Mapper ($f_{\text{Map}}$) is deep neural network which takes in the current RGB observation, $s_t \in \mathbb{R}^{1\times H\times W}$, the current pose of the agent $x_t$, and the map at the previous time step $m_{t-1} \in \mathbb{R}^{2\times M\times M}$ and outputs an updated map, $m_t \in \mathbb{R}^{2\times M\times M}$ (see Figure 2).

$$m_t = f_{\text{Map}}(s_t, x_t, m_{t-1} | \theta_M)$$

where $\theta_M$ denote the parameters of the Mapper. The Mapper can be broken down into two parts, a Projection Unit ($f_P$) and a Map Aggregation Unit ($f_A$). The Projection Unit outputs a egocentric top-down 2D spatial map, $p_t \in [0, 1]^{2\times V\times V}$ (where $V = 64$ is the vision range), predicting the obstacles and the explored area in the current observation: $p_t = f_P(s_t | \theta_P)$, where $\theta_P$ are the parameters of the Projection Unit. It consists of Resnet18 convolutional layers to produce an embedding of the observation. This embedding is passed through two fully-connected layers followed by 3 deconvolutional layers to get the first-person top-down 2D spatial map prediction.

The Map Aggregation Unit ($f_A$) transforms the egocentric spatial map prediction to the geocentric frame using the current pose of the agent ($x_t$) using Spatial Transformation [21] and aggregated with the previous spatial map ($m_{t-1}$) using Channel-wise Pooling operation: $m_t = f_A(p_t, x_t, m_{t-1} | \theta_A)$, where $\theta_A$ denotes the parameters of the Map Aggregation Unit. Note that the parameters of the Spatial Transformation are not learnt but calculated using the pose of the agent.

Combining both the Projection and Map Aggregation Unit:

$$m_t = f_{\text{Map}}(s_t, x_t, m_{t-1} | \theta_M) = f_A(f_P(s_t | \theta_P), x_t, m_{t-1} | \theta_A) \quad \text{where} \quad \theta_M = [\theta_P, \theta_A]$$

Global Policy. The Global Policy takes the Task $T_i$ and $h_t \in [0, 1]^{4\times M\times M}$ as input, where the first two channels of $h_t$ are the spatial map $m_t$ given by the Mapper, the third channel represents the current agent position estimated by the Mapper, the fourth channel represents the visited locations, i.e. $\forall i, j \in \{1, 2, \ldots, M\}$:

$$h_t[c, i, j] = m_t[c, i, j] \quad \forall c \in \{0, 1\}$$
$$h_t[2, i, j] = 1 \quad \text{if } i = x_t[0] \text{ and } j = x_t[1]$$
$$h_t[3, i, j] = 1 \quad \text{if } (i, j) \in [(x_k[0], x_k[1])]_{k \in \{0, 1, \ldots, t\}}$$

The input $h_t$ is first resized into $4 \times G \times G$ using bilinear upsampling or downsampling. The Global Policy uses a 5-layer convolutional neural network to predict a long-term goal, $g_t^l$ in $G \times G$ space: $g_t^l = \pi G(h_t | \theta_G, x_t)$, where $\theta_G, x_t$ are the parameters of the Global Policy specific to the task $T_i$.

Planner. The Planner takes the long-term goal ($g_t^l$) and the spatial obstacle map ($m_t$) as input and computes the short-term goal $g_t^s$, i.e. $g_t^s = f_{\text{Plan}}(g_t^l, m_t)$. It computes the shortest path from the current agent location to the long-term goal ($g_t^l$) using the Fast Marching Method [41] based on the current spatial map $m_t$. The unexplored area is considered as free space for planning. We compute a short-term goal coordinate (farthest point within $d_s (= 0.25 m)$ from the agent) on the planned path. We transform the short-term goal coordinate into relative distance and angle from the agent’s location and pass it to the Local Policy.
Local Policy. The Local Policy takes as input the current RGB observation \(s_t\) and the short-term goal \(g_t\) and outputs a navigational action, \(a_t = \pi_L(s_t, g_t; \theta_L)\), where \(\theta_L\) are the parameters of the Local Policy. It consists of a 3-layer convolutional neural network followed by a GRU layer. See Figure 3 for the information flow between the Global policy, the Planner and the Local policy.

4 Experimental setup

We use the Habitat simulator \[39\] with the Gibson \[45\] and Matterport (MP3D) \[4\] datasets for our experiments. Both Gibson and Matterport consist of scenes which are 3D reconstructions of real-world environments, however Gibson is collected using a different set of cameras, consists mostly of office spaces while Matterport consists of mostly homes with a larger average scene area. We will use Gibson as our training domain, and use Matterport for domain generalization experiments. The observation space consists of RGB images of size \(3 \times 128 \times 128\) and the actions space consists of four actions: move\_forward, turn\_left, turn\_right, stop. We consider two tasks in both the domains, PointGoal \[38, 1, 39\] and Exploration \[6\].

PointGoal. PointGoal has been the most studied task in recent literature on navigation where the objective is to navigate to a goal location whose relative coordinates are given as input in a limited time budget. We follow the PointGoal task setup from Savva et al. \[39\], using train/val/test splits for both Gibson and Matterport datasets. Note that the set of scenes used in each split is disjoint, which means the agent is tested on new scenes never seen during training. Gibson test set is not public but rather held out on an online evaluation server\[7\]. We report the performance of our model on the Gibson test set when submitted to the online server but also use the validation set as another test set for extensive comparison and analysis. We do not use the validation set for hyper-parameter tuning.

Savva et al. \[39\] identify two measures to quantify the difficulty of a PointGoal dataset. The first is the average geodesic distance (distance along the shortest path) to the goal location from the starting location of the agent, and the second is the average geodesic to Euclidean distance ratio (GED ratio). The GED ratio is always greater than or equal to 1, with higher ratio resulting in harder episodes. The train/val/test splits in Gibson dataset come from the same distribution of having similar average geodesic distance and GED ratio. In order to analyze the performance of the proposed model on out-of-set goal distribution, we create two harder sets, Hard-Dist and Hard-GEDR. In the Hard-Dist set, the geodesic distance to goal is always more than 10m and the average geodesic distance to the goal is 13.48m as compared to 6.9/6.5/7.0m in train/val/test splits \[39\]. Hard-GEDR set consists of episodes with an average GED ratio of 2.52 and a minimum GED ratio of 2.0 as compared to average GED ratio 1.37 in the Gibson val set.

We also follow the episode specification from Savva et al. \[39\]. Each episode ends when either the agent takes the stop action or at a maximum of 500 timesteps. An episode is considered a success when the final position of the agent is within 0.2m of the goal location. In addition to Success rate (Succ), we also use Success weighted by (normalized inverse) Path Length or SPL as a metric for evaluation for the PointGoal task as proposed by Anderson et al. \[1\].

Exploration. We follow the Exploration task setup proposed by Chen et al. \[6\] where the objective to maximize the coverage in a fixed time budget. Chen et al. \[6\] define coverage as the total area in the map known to be either traversable or non-traversable. We construct train/val/test sets of the Exploration task in the Habitat Simulator by simply taking the agent starting locations from the train/val/test splits for both Gibson and Matterport provided by Savva et al. \[39\] for the PointGoal task. This maintains the same split of train/val/test scenes for both the domains across both the tasks. The episode specification for the Exploration task in \[6\] also allows for a fixed time budget of 500 timesteps. The evaluation metric is coverage area in \(m^2\).

4.1 Training Details

We train our model for the Exploration task in the Gibson domain and transfer it to the PointGoal task and the Matterport domain. The Projection Unit in the mapper consists of ResNet18 convolutional layers followed by 2 fully-connected layers trained with dropout of 0.5, followed by 3 deconvolutional layers. The Planner uses the deterministic Fast Marching Method and is not trained. The Global Policy is a 5 layer fully-convolutional network, while the Local Policy consists of a 3-layer Convolutional network followed by a GRU. The Mapper is trained to predict egocentric projections using supervised learning.
Table 1: Exploration performance of the proposed model, Active Neural Mapping (ANM) and baselines.

<table>
<thead>
<tr>
<th>Method</th>
<th>Gibson Val Coverage ($m^2$)</th>
<th>MP3D Test Coverage ($m^2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>11.52</td>
<td>25.92</td>
</tr>
<tr>
<td>RL + 3LConv + GRU</td>
<td>21.60</td>
<td>33.55</td>
</tr>
<tr>
<td>RL + Res18 + GRU</td>
<td>24.48</td>
<td>33.12</td>
</tr>
<tr>
<td>RL + Res18 + GRU + AuxDepth</td>
<td>28.80</td>
<td>45.36</td>
</tr>
<tr>
<td>RL + Res18 + GRU + ProjDepth</td>
<td>30.24</td>
<td>41.04</td>
</tr>
<tr>
<td>ANM</td>
<td><strong>43.20</strong></td>
<td><strong>63.07</strong></td>
</tr>
</tbody>
</table>

learning. The ground truth egocentric projection is computed using geometric projections from ground truth depth. The egocentric to geocentric conversion is deterministic based on the pose of the agent. The Global and Local policies are both trained using Reinforcement Learning. The reward for the Global policy is the increase in coverage and the reward for the Local policy is the reduction in Euclidean distance to the short-term goal. All the modules are trained simultaneously. Their parameters are independent, but the data distribution is inter-dependent. Based on the actions taken by the Local policy, the future input to Mapper changes, which in turn changes the map input to the Global policy and consequently affects the short-term goal given to the Local policy. For more architecture and hyperparameter details please refer to the supplementary material and the code. We will also open-source the code.

In many prior works on the PointGoal and Exploration tasks as well as in the Habitat Challenge, pose is assumed to be known explicitly or implicitly (relative pose to a reference point) [38, 14, 6, 39]. Even if the pose is not known, the agent pose is easy to estimate due to deterministic actions with fixed translations and rotations in the Habitat simulator. Consequently, due to lack of realistic pose noise in the simulator and for a fair comparison with prior work, we assume access to the transition function ($f_t$) which gives the agent pose.

4.2 Baselines

We use a range of end-to-end learning methods trained using Reinforcement Learning (RL) and Imitation Learning (IL) as baselines:

RL + Blind: A policy which only receives PointGoal as input trained with RL.

RL + 3LConv + GRU: An RL Policy with 3 layer convolutional network followed by a GRU [7] as described by Savva et al. [39] which is also identical to our Local Policy architecture.


RL + Res18 + GRU + AuxDepth: This baseline is adapted from Mirowski et al. [29] who use depth prediction as an auxiliary task. We use the same architecture as our Mapper (conv layers from ResNet18) with one additional deconvolutional layer for Depth prediction followed by 3 layer convolution and GRU for the policy.

RL + Res18 + GRU + ProjDepth: This baseline is adapted from Chen et al. [6] who project the depth image in an egocentric top-down in addition to the RGB image as input to the RL policy. Since we do not have depth as input, we use the architecture from RL + Res18 + GRU + AuxDepth for depth prediction and project the predicted depth before passing to 3Layer Conv and GRU policy.


Cognitive Mapping and Planning (CMP) [14] is an end-to-end learning model with differentiable components for mapping and planning. It is trained using imitation learning.

All RL baselines are trained using PPO [40] and IL baselines are trained using DAGGER [34]. We use increase in coverage as the reward for RL policies for the Exploration task. For the PointGoal task, we use reduction in the geodesic distance as the reward for RL baselines, and shortest path actions as supervision for the IL baselines. Note that both the RL and IL baselines require the ground truth map to compute the geodesic distance to goal (RL) or to compute the shortest path (IL) during training. In contrast, our method does not require ground truth map during training time as the Local Policy is trained with reduction in the Euclidean distance to the short-term goal as the reward.
Table 2: Performance of the proposed model, ANM and all the baselines on the Exploration task. ’ANM - Task Transfer’ refers to the ANM model transferred to the PointGoal task after training on the Exploration task.

<table>
<thead>
<tr>
<th>Test Setting →</th>
<th>Gibson Val</th>
<th>MP3D Test</th>
<th>Hard-GEDR</th>
<th>Hard-Dist</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Train Task</strong></td>
<td><strong>Method</strong></td>
<td>Succ</td>
<td>SPL</td>
<td>Succ</td>
</tr>
<tr>
<td>PointGoal</td>
<td>Random</td>
<td>0.027</td>
<td>0.021</td>
<td>0.010</td>
</tr>
<tr>
<td></td>
<td>RL + Blind</td>
<td>0.625</td>
<td>0.421</td>
<td>0.136</td>
</tr>
<tr>
<td></td>
<td>RL + 3LConv + GRU</td>
<td>0.550</td>
<td>0.406</td>
<td>0.102</td>
</tr>
<tr>
<td></td>
<td>RL + Res18 + GRU</td>
<td>0.561</td>
<td>0.422</td>
<td>0.160</td>
</tr>
<tr>
<td></td>
<td>RL + Res18 + GRU + AuxDepth</td>
<td>0.640</td>
<td>0.461</td>
<td>0.189</td>
</tr>
<tr>
<td></td>
<td>RL + Res18 + GRU + ProjDepth</td>
<td>0.614</td>
<td>0.436</td>
<td>0.134</td>
</tr>
<tr>
<td></td>
<td>IL + Res18 + GRU</td>
<td>0.716</td>
<td>0.673</td>
<td>0.221</td>
</tr>
<tr>
<td>CMP</td>
<td>0.738</td>
<td>0.683</td>
<td>0.237</td>
<td>0.219</td>
</tr>
<tr>
<td>ANM</td>
<td>0.951</td>
<td>0.848</td>
<td>0.593</td>
<td>0.496</td>
</tr>
<tr>
<td>Exploration ANM - Task Transfer</td>
<td>0.950</td>
<td>0.846</td>
<td>0.588</td>
<td>0.490</td>
</tr>
</tbody>
</table>

5 Results

Exploration. We first train the proposed ANM model and all the exploration baselines for the Exploration task with 10 million frames on the Gibson training set. The results are shown in Table 1. The proposed model achieves an average coverage of 43.20m² as compared to 30.24m² for the best baseline. This indicates that the proposed model is more efficient and effective at exhaustive exploration as compared to end-to-end learning methods. This is because our hierarchical policy architecture reduces the horizon of the long-term exploration problem as instead of taking tens of low-level navigational actions, the Global policy only takes few long-term goal actions. We also report the domain generalization performance on the Exploration task in Table 1 (see shaded region), where all models trained on Gibson are evaluated on the Matterport domain. ANM leads to higher domain generalization performance (63.07m² vs 45.36m²). The absolute coverage is higher for the Matterport domain as it consists on larger scenes on average.

Task Generalization: PointGoal. The Local policy is only trained to reach the short-term goal. This makes the Local policy task-invariant, as the Global policy can select different long-term goals based on the task and Local policy can be transferred across tasks without any fine-tuning. We can adapt the policy trained for the Exploration task above to the PointGoal task without any additional data. We just fix the Global policy to always output the PointGoal coordinates as the long-term goal and use the Local and Mapper trained for the Exploration task. In Table 2, we show the performance of the proposed model transferred to the PointGoal task along with the baselines trained on the PointGoal task with the same amount of data (10 million frames). The proposed model achieves a success rate/SPL of 0.950/0.846 as compared to 0.738/0.683 for the best baseline model on Gibson val set. We also report the performance of the proposed model trained from scratch on the PointGoal task for 10 million frames. The results indicate that the performance of ANM transferred from Exploration is comparable to ANM trained on PointGoal. This highlights a key advantage of our model that it allows us to transfer the knowledge of obstacle avoidance and control in low-level navigation across tasks, as the Local Policy and Mapper are task-invariant.

We also evaluated ANM on the private Gibson test-std dataset by submitting our model to the online server. ANM achieves an SPL of 0.79 on Gibson test-std set as compared to 0.47 (RGB-RL-PPO), 0.40 (BlindRLPPO), 0.23 (GoalFollower), 0.02 (Random) for the baselines. Note that these RL baselines were trained for 75 million frames [39] as compared to 10 million for ANM.

Sample efficiency. RL models are typically trained for more than 10 million samples. In order to compare the performance and sample-efficiency, we trained the best performing RL model (RL + Res18 + GRU + ProjDepth) for 75 million frames and it achieved a Succ/SPL of 0.678/0.486. The best performing imitation learning-based baseline CMP [14] gets a Succ/SPL of 0.738/0.683 at 10 million frames as shown in Table 2. ANM reaches the performance of 0.789/0.703 SPL/Succ at 1 million frames. These numbers indicate that ANM achieves > 70x speedup as compared to the best RL baseline and > 10x speedup as compared to the best IL baseline in terms of sample efficiency.
Figure 4: Performance of the proposed ANM model along with CMP and IL + Res18 + GRU (GRU) baselines with increase in geodesic distance to goal and increase in GED Ratio on the Gibson Val set.

Successful Trajectories

Failure Case

Figure 5: Figure showing sample trajectories of the proposed model along with predicted map in the PointGoal task. The starting and goal locations are shown by black squares and blue circles, respectively. Ground truth map is under-laid in grey. Map prediction is overlaid in green, with dark green denoting correct predictions and light green denoting false positives. Blue shaded region shows the explored area prediction. On the left, we show some successful trajectories which indicate that the model is effective at long distance goals with high GED ratio. On the right, we show a failure case due to mapping error.

Domain and Goal Generalization: In Table 2 (see shaded region), we evaluate all the baselines and ANM trained on the PointGoal task in the Gibson domain on the test set in Matterport domain as well as the harder goal sets in Gibson. We also transfer ANM trained on Exploration in Gibson on all the 3 sets. The results show that ANM outperforms all the baselines at all generalization sets. Interestingly, RL based methods almost fail completely on the Hard-Dist set. We also analyze the performance of the proposed model as compared to two best baselines CMP and IL + Res18 + GRU as a function of geodesic distance to goal and GED ratio in Figure 4. The performance of the baselines drops faster as compared to ANM, especially with increase in goal distance. This indicates that end-to-end learning methods are effective at short-term navigation but struggle when long-term planning is required to reach a distant goal. In Figure 5, we show some example trajectories of the ANM model along with the predicted map. The successful trajectories indicate that the model exhibits strong backtracking behavior which makes it effective at distant goals requiring long-term planning.

We believe the strong generalization performance and improvement in sample efficiency are partly due to using a deterministic planning algorithm as compared to learning to plan. The Planner acts as a natural link between the global and local policy. In order to train models end-to-end, prior methods, such as CMP [14], use differentiable trainable planners such as Value Iteration Networks [42] or Gated Path Planning Networks [28] which have suboptimal accuracy and degrade in performance as the size of the map increases [42]. However, since our Global policy outputs the long-term goal as a discrete action, we can use classical deterministic planning algorithms such as Fast Marching Method [41] which are perfect at planning.

6 Conclusion

In this paper, we proposed a modular navigational model which leverages the strengths of classical and learning-based navigational methods. We show that the proposed model outperforms prior methods on both Exploration and PointGoal tasks and shows strong generalization across domains, goals, and tasks. In future, the proposed model can be extended to complex semantic tasks such as Semantic Goal Navigation and Embodied Question Answering by using a semantic Mapper which creates multi-channel map capturing semantic properties of the objects in the environment. The model can also be combined with prior work on Localization to relocalize in a previously created map for efficient navigation in subsequent episodes.

4See https://sites.google.com/view/active-neural-mapping for visualization videos.
References


