LEARNING TO EXPLORE USING ACTIVE NEURAL MAPPING

Anonymous authors
Paper under double-blind review

ABSTRACT

This work presents a modular and hierarchical approach to learn policies for exploring 3D environments. Our approach leverages the strengths of both classical and learning-based methods, by using analytical path planners with learned mappers, and global and local policies. Use of learning provides flexibility with respect to input modalities (in mapper), leverages structural regularities of the world (in global policies), and provides robustness to errors in state estimation (in local policies). Such use of learning within each module retains its benefits, while at the same time, hierarchical decomposition and modular training allow us to sidestep the high sample complexities associated with training end-to-end policies. Our experiments in *visually* and *physically* realistic simulated 3D environments demonstrate the effectiveness of our approach over past learning and geometry-based approaches.[1]

1 INTRODUCTION

Navigation is a critical task in building intelligent agents. Navigation tasks can be expressed in many forms, for example, point goal tasks involve navigating to a specific coordinates and semantic navigation involves finding path to a specific scene or object. Such tasks may need to be performed in known (already mapped) or unknown environments. Irrespective of the task or the setting, a core problem in navigation is exploration, *i.e.*, how to efficiently visit as much of the environment. This is useful for pre-mapping in known environments, or actually solving tasks in known environments.

Recent work from Chen et al. (2019) has used end-to-end learning to tackle this problem. Their motivation is three fold: *a)* learning provides flexibility to the choice of input modalities (classical systems rely on observing geometry through use of specialized sensors, while learning systems can infer geometry directly from RGB images), *b)* use of learning can improve robustness to errors in explicit state estimation, and *c)* learning can effectively leverage structural regularities of the real world, leading to more efficient behavior in previously unseen environments. This lead to their design of an end-to-end trained neural network based policy that processed raw sensory observations to directly output actions that the agent should execute.

While use of learning for exploration is well motivated, casting the exploration problem as an end-to-end learning problem has its own drawbacks. Learning about mapping, state-estimation and path-planning purely from data in an end-to-end manner can be prohibitively expensive. Consequently, past end-to-end learning work for exploration from Chen et al. (2019) relies on use of imitation learning and many millions of frames of experience, but still performs worse than classical methods that don't require any training at all.

This motivates our work. In this paper, we investigate alternate formulations of employing learning for exploration that retains the advantages that learning has to offer, but doesn't suffer from the drawbacks of full-blown end-to-end learning. Our key conceptual insight is that use of learning for leveraging structural regularities of indoor environments, robustness to state-estimation errors, and flexibility with respect to input modalities, happens at different time scales and can thus be factored out. This motivates use of learning in a modular and hierarchical fashion inside of what one may call a 'classical navigation pipeline'. This results in navigation policies that can work with raw sensory inputs such as RGB images, are robust to state estimation errors, and leverage regularities of real world layout. This results in extremely competitive performance over both geometry-based methods and recent learning-based methods; at the same time requiring a fraction of the number of samples.

[1]See `https://sites.google.com/view/active-neural-mapping` for visualization videos.

More specifically, our proposed exploration architecture comprises of a learned mapper (and pose estimator), a global policy, and a local policy, that are interfaced via the map and an analytical path planner. The learned mapper, together with the pose estimator, produces free space maps from input RGB images. The global policy consumes this free-space map and employs learning to exploit structural regularities in layout of real world environments to produce long-term goals. These long-term goals are used to generate short-term goals for the local policy (using a geometric path-planner). This local policy uses learning to directly map raw RGB images to actions that the agent should execute. Use of learning in mapper provides flexibility with respect to input modality, learned global policy can exploit regularities in layout of real world layout of environments, while learned local policies can use visual feedback to exhibit more robust behaviour. At the same time, hierarchical and modular design and use of analytical planning, significantly cuts down the search space during training, leading to better performance as well as sample efficient learning.

We demonstrate our proposed approach in *visually* and *physically* realistic simulators for the task of geometric exploration (visit as much area as possible). We work with the Habitat simulator from Savva et al. (2019). While Habitat is already visually realistic (it uses real world scans from Chang et al. (2017); Xia et al. (2018) as environments), we improve its physical realism by using actuation and odometry sensor noise models, that we collected by conducting physical experiments on a real mobile robot. Our experiments and ablations in this realistic simulation reveal the effectiveness of our proposed approach for the task of exploration. A straight-forward modification of our method also tackles point-goal navigation tasks, and won the AI Habitat challenge at CVPR2019 across all tracks.

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.

Navigation Approaches. Classical approaches to navigation break the problem into two parts: mapping and path planning. Mapping is done via simultaneous localization and mapping (Thrun et al., 2005; Hartley and Zisserman, 2003; Fuentes-Pacheco et al., 2015), by fusing information from multiple views of the environment. While sparse reconstruction can be done well with monocular RGB images (Mur-Artal and Tardós, 2017), dense mapping is inefficient (Newcombe et al., 2011) or requires specialized scanners such as Kinect (Izadi et al., 2011). Maps are used to compute paths to goal locations via path planning (Kavraki et al., 1996; Lavalle and Kuffner Jr, 2000; Canny, 1988). These classical methods have inspired recent learning-based techniques. Researchers have designed neural network policies that reason via spatial representations (Gupta et al., 2017; Parisotto and Salakhutdinov, 2018; Zhang et al., 2017; Henriques and Vedaldi, 2018; Gordon et al., 2018), topological representations (Savinov et al., 2018a;b), or use differentiable and trainable planners (Tamar et al., 2016; Lee et al., 2018; Gupta et al., 2017; Khan et al., 2017). Our work furthers this research, and we study a hierarchical and modular decomposition of the problem, and employ learning inside these components instead of end-to-end learning. Research also focuses on incorporating semantics in SLAM (Pronobis and Jensfelt, 2012; Walter et al., 2013).

Exploration in Navigation. While a number of works focus on passive map-building, path planning and goal-driven policy learning, a much smaller body of work tackles the problem of active SLAM, i.e., how to actively control the camera for map building. We point readers to Fuentes-Pacheco et al. (2015) for a detailed survey, and summarize major these below. Most such works frame this problem as a Partially Observable Markov Decision Process (POMDP) that are approximately solved (Martinez-Cantin et al., 2009; Kollar and Roy, 2008), and or seek to find a sequence of actions that minimizes uncertainty of maps (Stachniss et al., 2005; Carlone et al., 2014). Another line of work, explores by picking vantage points (such as on the frontier between explored and unexplored regions (Dornhege and Kleiner, 2013; Holz et al., 2010; Yamauchi, 1997; Xu et al., 2017)). Recent works from Chen et al. (2019); Savinov et al. (2018b); Fang et al. (2019) attack this problem via learning. Our proposed modular policies unify the last two lines of research, and we show improvements over representative methods from both these lines of work. Exploration has also been studied more generally in RL for faster training (Schmidhuber, 1991).

Hierarchical and Modular Policies. Hierarchical RL (Barto and Mahadevan, 2003) is an active area of research, aimed at automatically discovering hierarchies to speed up learning. However, this has proven to be challenging, and thus most work has resorted to using hand-defining hierarchies. For example in context of navigation, Bansal et al. (2019) and Kaufmann et al. (2019) design modular policies for navigation, that interface learned policies with low-level feedback controllers.

3 Task Setup

We follow the exploration task setup proposed by Chen et al. 2019 where the objective is to maximize the coverage in a fixed time budget. The coverage is defined as the total area in the map known to be traversable. Our objective is train a policy which takes in an observation s_t at each time step t and outputs a navigational action a_t to maximize the coverage.

We try to make our experimental setup in simulation as realistic as possible with the goal of transferring trained policies to the real world. We use the Habitat simulator (Savva et al., 2019) with the Gibson (Xia et al., 2018) and Matterport (MP3D) (Chang et al., 2017) datasets for our experiments. Both Gibson and Matterport datasets are based on real-world scene reconstructions are thus significantly more realistic than synthetic SUNCG dataset used for past research on exploration (Chen et al., 2019; Fang et al., 2019).

In addition to synthetic scenes, prior works on learning-based navigation have also assumed simplistic agent motion. Some works limit agent motion on a grid with 90 degree rotations (Zhu et al., 2017; Gupta et al., 2017; Parisotto and Salakhutdinov, 2018; Chaplot et al., 2018). Other works which implement fine-grained control, typically assume unrealistic agent motion without any noise (Savva et al., 2019; Fang et al., 2019). This consequently leads to another unrealistic assumption of knowledge of perfect agent pose. This is because since the motion is simplistic, it becomes trivial to estimate the agent pose in most cases even if it is not assumed to be known. The reason behind these assumptions on agent motion and pose is that motion and sensor noise models are not known. In order to relax both these assumptions, we collect motion and sensor data in the real-world and implement more realistic agent motion and sensor noise models in the simulator as described in the following subsection.

3.1 Actuation and Sensor Noise Model

We represent the agent pose by (x, y, o) where x and y represent the xy co-ordinate of the agent measured in metres and o represents the orientation of the agent in radians (measured counter-clockwise from x-axis). Without loss of generality, assume agents starts at $p_0 = (0, 0, 0)$. Now, suppose the agent takes an action a_t. Each action is implemented as a control command on a robot. Let the corresponding control command be $\Delta u_a = (x_a, y_a, o_a)$. Let the agent pose after the action be $p_1 = (x^\star, y^\star, o^\star)$. The actuation noise ($\epsilon_{act}$) is the difference between the actual agent pose (p_1) after the action and the intended agent pose ($p_0 + \Delta u$):

$$\epsilon_{act} = p_1 - (p_0 + \Delta u) = (x^\star - x_a, y^\star - y_a, o^\star - o_a)$$

Mobile robots typically have sensors which estimate the robot pose as it moves. Let the sensor estimate of the agent pose after the action be $p_1' = (x', y', o')$. The sensor noise (ϵ_{sen}) is given by the difference between the sensor pose estimate (p_1') and the actual agent pose(p_1):

$$\epsilon_{sen} = p_1' - p_1 = (x' - x^\star, y' - y^\star, o' - o^\star)$$

In order to implement the actuation and sensor noise models, we would like to collect data for navigational actions in the Habitat simulator. We use three default navigational actions: Forward: move forward by 25cm, Turn Right: on the spot rotation clockwise by 10 degrees, and Turn Left: on the spot rotation counter-clockwise by 10 degrees. The control commands are implemented as $u_{Forward} = (0.25, 0, 0)$, $u_{Right} : (0, 0, -10 * \pi/180)$ and $u_{Left} : (0, 0, 10 * \pi/180)$. In practice, a robot can also rotate slightly while moving forward and translate a bit while rotating on-the-spot, creating rotational actuation noise in forward action and similarly, a translation actuation noise in on-the-spot rotation actions.

We use a LoCoBot[2] to collect data for building the actuation and sensor noise models. We use the pyrobot API (Murali et al., 2019) along with ROS (Quigley et al., 2009) to implement the control commands and get sensor readings. For each action a, we fit a separate Gaussian Mixture Model for the actuation noise and sensor noise, making a total of 6 models. Each component in these Gaussian mixture models is a multi-variate Gaussian in 3 variables, x, y and o. For each model, we

[2]http://locobot.org

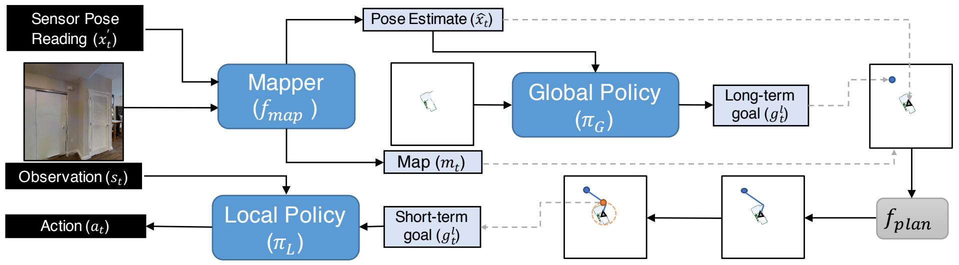


Figure 1: Overview of our approach. We use a neural network based Mapper that predicts a map and agent pose estimate from incoming RGB observations and sensor readings. This map is used by a Global policy to output a long-term goal, which is converted to a short-term goal using an analytic path planner. A Local Policy is trained to navigate to this short-term goal.

collect 600 datapoints. The number of components in each Gaussian mixture model are chosen using cross-validation. We implement these actuation and sensor noise models in the Habitat simulator for our experiments. We will open-source the collected data and the noise models, along with their implementation in the Habitat simulator.

4 METHODS

We propose a modular navigation model, 'Active Neural Mapping'. It consists of three components: a *Mapper*, a *Global policy* and a *Local policy*. The Mapper predicts the map of the environment and estimates the pose of the agent based on the current observations and previous estimates. The Global policy uses the predicted map and agent pose to produce a long-term goal. The long-term goal is converted into a short-term goal using path planning. The Local policy takes navigational actions based on the current observation to reach the short-term goal. See Figure 1 for an overview.

Map Representation. 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 $25cm^2$ ($5cm \times 5cm$) 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, $x_0 = (M/2, M/2, 0.0)$.

Mapper. The Mapper (f_{Map}) takes in the current RGB observation, s_t, the current and last sensor reading of the agent pose $x'_{t-1:t}$, last agent pose and map estimates, $\hat{x}_{t-1}, m_{t-1}$ and outputs an updated map, m_t, and the current agent pose estimate, $\hat{x}_t$, (see Figure 2): $m_t, \hat{x}_t = f_{Map}(s_t, x'_{t-1:t}, \hat{x}_{t-1}, m_{t-1}|\theta_M)$, where θ_M denote the trainable parameters of the Mapper. It consists of two learned components, a Projection Unit and a Pose Estimator. The Projection Unit outputs a egocentric top-down 2D spatial map, $p^{ego}_t \in [0,1]^{2\times V\times V}$ (where V is the vision range), predicting the obstacles and the explored area in the current observation. The Pose Estimator predicts the agent pose ($\hat{x}_t$) based on past pose estimate ($\hat{x}_{t-1}$) and last two egocentric map predictions ($p^{ego}_{t-1:t}$). It essentially compares the current egocentric map prediction to the last egocentric map prediction transformed to the current frame to predict the pose change between the two maps. The egocentric map from the Projection Unit is transformed to a geocentric map based on the pose estimate given by the Pose Estimator and then aggregated with the previous spatial map (m_{t-1}) to get the current map(m_t). More implementation details of the Mapper are provided in the Appendix.

Global Policy. The Global Policy takes $h_t \in [0,1]^{4\times M\times M}$ 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. $\forall i, j \in \{1, 2, \ldots, m\}$:

$$\begin{aligned}
h_t[c,i,j] &= m_t[c,i,j] && \forall c \in \{0,1\} \\
h_t[2,i,j] &= 1 && \text{if } i = \hat{x}_t[0] \text{ and } j = \hat{x}_t[1] \\
h_t[3,i,j] &= 1 && \text{if } (i,j) \in [(\hat{x}_k[0], \hat{x}_k[1])]_{k\in\{0,1,\ldots,t\}}
\end{aligned}$$

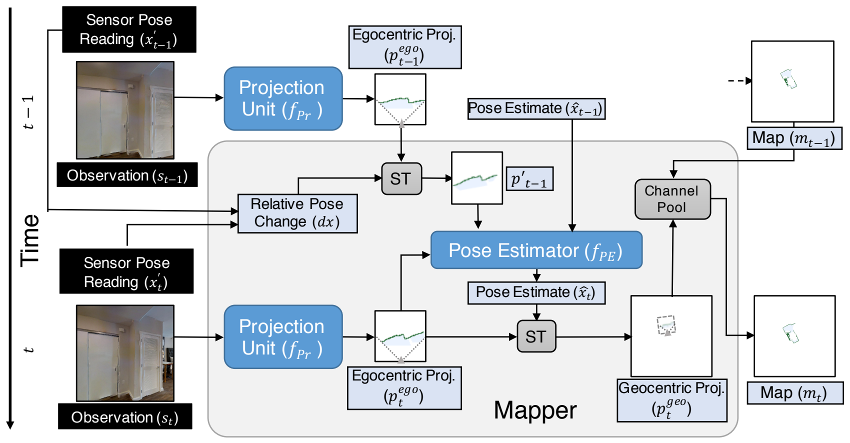

Time
$t-1$
t
Sensor Pose Reading (x'_{t-1})
Observation (s_{t-1})
Sensor Pose Reading (x'_t)
Observation (s_t)
Projection Unit (f_{Pr})
Egocentric Proj. (p^{ego}_{t-1})
Egocentric Proj. (p^{ego}_t)
Pose Estimate ($\hat{x}_{t-1}$)
ST
p'_{t-1}
Relative Pose Change (dx)
Pose Estimator (f_{PE})
Pose Estimate ($\hat{x}_t$)
Channel Pool
Map (m_{t-1})
Map (m_t)
Geocentric Proj. (p^{geo}_t)
Mapper

Figure 2: Architecture of the mapper: The Mapper (f_{Map}) takes in the current RGB observation, s_t, the current and last sensor reading of the agent pose $x'_{t-1:t}$, last agent pose estimate, $\hat{x}_{t-1}$ and the map at the previous time step m_{t-1} and outputs an updated map, m_t and the current agent pose estimate, $\hat{x}_t$. 'ST' denotes spatial transformation.

We perform two transformations before passing h_t to the Global Policy model. The first transformation subsamples a window of size $4 \times G \times G$ around the agent from h_t. The second transformation performs max pooling operations to get an output of size $4 \times G \times G$ from h_t. Both the transformations are stacked to form a tensor of size $8 \times G \times G$ and passed as input to the Global Policy model. The Global Policy uses a 5-layer convolutional neural network to predict a long-term goal, g^l_t in $G \times G$ space: $g^l_t = \pi_G(h_t|\theta_G)$, where θ_G are the parameters of the Global Policy.

Planner. The Planner takes the long-term goal (g^l_t), the spatial obstacle map (m_t) and the agnet pose estimate ($\hat{x}_t$) as input and computes the short-term goal g^s_t, i.e. $g^s_t = f_{Plan}(g^l_t, m_t, \hat{x}_t)$. It computes the shortest path from the current agent location to the long-term goal (g^l_t) using the Fast Marching Method (Sethian, 1996) 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 $d_s(= 1.25m)$ from the agent) on the planned path.

Local Policy. The Local Policy takes as input the current RGB observation (s_t) and the short-term goal (g^s_t) and outputs a navigational action, $a_t = \pi_L(s_t, g^s_t|\theta_L)$, where θ_L are the parameters of the Local Policy. The short-term goal coordinate is transformed into relative distance and angle from the agent's location before being passed to the Local Policy. It consists of a 3-layer convolutional neural network followed by a GRU layer.

5 Experimental Setup

We use the Habitat simulator (Savva et al., 2019) with the Gibson (Xia et al., 2018) and Matterport (MP3D) (Chang et al., 2017) 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 base odometry sensor readings of size 3×1 denoting the change in agent's x-y coordinates and orientation. The actions space consists of three actions: `move_forward`, `turn_left`, `turn_right`. Both the base odometry sensor readings and the agent motion based on the actions are noisy. They are implemented using the sensor and actuation noise models based on real-world data as discussed in Section 3.1.

We follow the Exploration task setup proposed by Chen et al. (2019) where the objective to maximize the coverage in a fixed time budget. Coverage is the total area in the map known to be traversable. We define a traversable point to be known if it is in the field-of-view of the agent and is less than $3m$ away. We use two evaluation metrics, the absolute coverage area in m^2 (**Cov**) and percentage of area explored in the scene (**% Cov**), i.e. ratio of coverage to maximum possible coverage in the corresponding scene. During training, each episode lasts for a fixed length of 1000 steps.

We use train/val/test splits provided by Savva et al. 2019 for both the 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 for the Pointgoal task. We use the validation as the test set for comparison and analysis for the Gibson domain. We do not use the validation set for hyper-parameter tuning. To analyze the performance of all the models with respect to the size of the scene, we split the Gibson validation set into two parts, a small set of 10 scenes with explorable area ranging from $16m^2$ to $36m^2$, and a large set of 4 scenes with traversable area ranging from $55m^2$ to $100m^2$. Note that the size of the map is usually much larger than the traversable area, with the largest map being about $23m$ long and $11m$ wide.

Training Details. We train our model for the Exploration task in the Gibson domain and transfer it to the Matterport domain. The Projection Unit is trained to predict egocentric projections, and the Pose Estimator is trained to predict agent pose using supervised learning. The ground truth egocentric projection is computed using geometric projections from ground truth depth. 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. We will also open-source the code.

Baselines. We use a range of end-to-end Reinforcement Learning (RL) methods as baselines:
RL + 3LConv: An RL Policy with 3 layer convolutional network followed by a GRU (Cho et al., 2014) as described by Savva et al. 2019 which is also identical to our Local Policy architecture.
RL + Res18: A RL Policy initialized with ResNet18 (He et al., 2016) pre-trained on ImageNet followed by a GRU.
RL + Res18 + AuxDepth: This baseline is adapted from Mirowski et al. 2017 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 + ProjDepth: This baseline is adapted form Chen et al. 2019 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 + AuxDepth for depth prediction and project the predicted depth before passing to 3Layer Conv and GRU policy.

For all the baselines, we also feed a 32-dimensional embedding of the sensor pose reading to the GRU along with the image-based representation. This embedding is also learnt end-to-end using RL. All baselines are trained using PPO (Schulman et al., 2017) with increase in coverage as the reward.

6 RESULTS

We train the proposed ANM model and all the baselines for the Exploration task with 10 million frames on the Gibson training set. The results are shown in Table 1. The results on the Gibson Val set are averaged over a total of 994 episodes in 14 different unseen scenes. The proposed model achieves an average absolute and relative coverage of $31.379m^2/0.924$ as compared to $24.958m^2/0.766$ for the best baseline. This indicates that the proposed model is more efficient and effective at exhaustive exploration as compared to the baselines. 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 ($57.228m^2/0.405$ vs $41.549m^2/0.297$). The abosulte coverage is higher for the Matterport domain as it consists of larger scenes on average. Some visualizations of policy executions are provided in the Appendix.[3]

In Fig. 3, we plot the relative coverage (% Cov) of all the models as the episode progresses on the large and small scene sets, as well as the overall Gibson Val set. The plot on the small scene set shows that ANM is able to almost completely explore the small scenes in around 500 steps, however the baselines are only able to explore 85% of the small scenes in 1000 steps (see Fig. 3 center). This

[3]See `https://sites.google.com/view/active-neural-mapping` for visualization videos.

,

Table 1: Exploration performance of the proposed model, Active Neural Mapping (ANM) and baselines. The baselines are adated from [1] Savva et al. 2019, [2] Mirowski et al. 2017 and [3] Chen et al. 2019.

	Gibson Val		Domain Generalization MP3D Test	
Method	% Cov.	Cov. (m2)	% Cov.	Cov. (m2)
RL + 3LConv [1]	0.72	22.382	0.297	41.277
RL + Res18	0.725	22.446	0.295	40.916
RL + Res18 + AuxDepth [2]	0.744	23.896	0.286	39.944
RL + Res18 + ProjDepth [3]	0.766	24.958	0.294	41.549
ANM	**0.924**	**31.379**	**0.405**	**57.228**

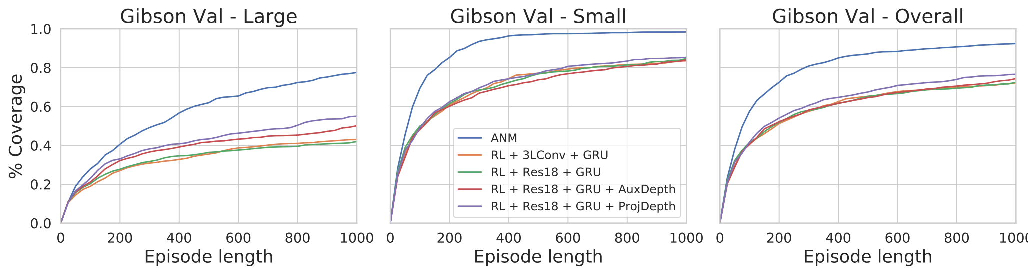


Figure 3: Plot showing the % Coverage as the episode progresses for ANM and the baselines on the large and small scenes in the Gibson Val set as well as the overall Gibson Val set.

indicates that ANM explores more efficiently in small scenes. The plot on the large scenes set shows that the performance gap between ANM and baselines widens as the episode progresses (see Fig. 3 left). Looking at the behaviour of the baselines, we saw that they often got stuck in local areas. This behaviour indicates that they are unable to remember explored areas over long-time horizons and are ineffective at long-term planning. On the other hand, ANM uses a Global policy on the map which allows it to have memory of explored areas over long-time horizons, and plan effectively to reach distant long-term goals by leveraging analytical planners. As a result, it is able to explore effectively in large scenes with long episode lengths.

6.1 Ablations

Local Policy. An alternative to learning a Local Policy is to have a deterministic policy which follows the plan given by the Planner. As shown in Table 2, the ANM model performs much worse without the Local Policy. The Local Policy is designed to adapt to small errors in Mapping. We observed Local policy overcoming both false positives and false negatives encountered in mapping. For example, the Mapper could sometime wrongly predict a carpet as an obstacle. In this case, the planner would plan to go around the carpet. However, if the short-term goal is beyond the carpet, the Local policy can understand that the carpet is not an obstacle based on the RGB observation and learn to walk over it. Similarly, we also observed cases where the Mapper didn't predict small obstacles very close to the agent as they were not in the field-of-view due to the height of the camera. In this case, the planner would plan a path through the obstacle where the deterministic policy would get stuck. Since the local policy is recurrent, it learns to navigate around these obstacles by getting feedback from the environment. When the policy tries to move forward but it can not, it gets feedback that there must be an obstacle.

Global Policy. An alternative to learning a Global Policy for sampling long-term goals is to use a classical algorithm called Frontier-based exploration (Yamauchi, 1997). A frontier is defined as the boundary between the explored free space and the unexplored space. Frontier-based exploration essentially sample points on this frontier as goals to explore the space. There are different variants of Frontier-based exploration based on the sampling strategy. Holz et al. (2010) compare different sampling strategies and find that sampling the point on the frontier closest to the agent gives the best results empirically. We implement this variant and replace it with our learned Global Policy. As shown in Table 2, Frontier-based exploration policy perform worse than the Global Policy. We

Table 2: Results of the ablation experiments on the Gibson environment.

Method	Gibson Val Overall % Cov.	Gibson Val Overall Cov. (m2)	Gibson Val Large % Cov.	Gibson Val Large Cov. (m2)	Gibson Val Small % Cov.	Gibson Val Small Cov. (m2)
ANM w/o Local Policy	0.882	29.673	0.713	45.810	0.950	23.219
ANM w/o Global Policy	0.879	29.242	0.678	44.443	0.960	23.161
ANM w/o Pose Estimation	0.889	29.964	0.724	47.585	0.955	22.916
ANM	**0.924**	**31.379**	**0.776**	**50.082**	**0.984**	**23.898**

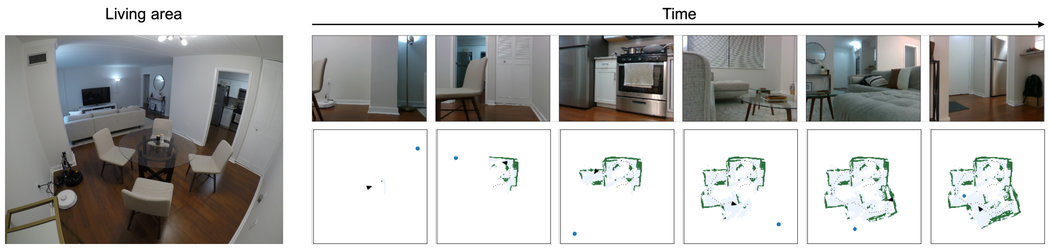


Figure 4: Real-world Transfer. Left: Image showing the living area in an apartment used for the real-world experiments. **Right:** Sample images seen by the robot and the predicted map. The long-term goal selected by the Global Policy is shown by a blue circle on the map.

observed that Frontier-based exploration spent a lot of time exploring corners or small area behind furniture. In contrast, the trained Global policy ignored small spaces and chose distant long-term goals which led to exploring more area.

Pose Estimation. A difference between ANM and the baselines is that ANM uses additional supervision to train the Pose Estimator. In order to understand whether the performance gain is coming from this additional supervision, we remove the Pose Estimator from ANM and just use the input sensor reading as our pose estimate. Results in Table 2 shows that the ANM still outperforms the baselines even without the Pose Estimator. We also tried passing the ground truth pose as input the baselines instead of the sensor reading. The performance of the baselines did not improve with the ground truth pose.

6.2 Real-world transfer

We deploy the trained ANM policy on a Locobot in the real-world. In order to match the real-world observations to the simulator observations as closely as possible, we change the simulator input configuration to match the camera intrinsics on the Locobot. This includes the camera height and horizontal and vertical field-of-views. In Figure 4, we show an episode of ANM exploring the living area in an apartment. The figure shows that the policy transfers well to the real-world and is able to effectively explore the environment. The long-term goals sampled by the Global policy (shown by blue circles on the map) are often towards open spaces in the explored map, which indicates that it is learning to exploit the structure in the map.[4]

6.3 Pointgoal Task Transfer.

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. In this task, 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), Success weighted by (normalized inverse) Path Length or SPL is also used as a metric for evaluation as proposed by Anderson et al. 2018.

All the baseline models trained for the task of Exploration either need to be retrained or atleast fine-tuned to be transferred to the Pointgoal task. The modularity of ANM provides it another advantage that it can be transferred to the Pointgoal task without any additional training. For transfer to the Pointgoal task, 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. We found that an ANM policy trained on exploration, when transferred to the Pointgoal task performed better than

[4]See `https://sites.google.com/view/active-neural-mapping` for videos of multiple real-world episodes.

several RL and Imitation Learning baselines trained on the Pointgoal task. The transferred ANM model achieves a success rate/SPL of 0.950/0.846 as compared to 0.827/0.730 for the best baseline model on Gibson val set. The ANM model also generalized significantly better than the baselines to harder goals and to the Matterport domain. In addition to better performance, ANM was also 10 to 75 times more sample efficient than the baselines. This transferred ANM policy was also the winner of the CVPR 2019 Habitat Pointgoal Navigation Challenge for both RGB and RGB-D tracks among over 150 submissions from 16 teams. These results highlight 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. More details about the Pointgoal experiments, baselines, results including domain and goal generalization on the Pointgoal task are provided in the supplementary material.

7 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.

References

Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

Somil Bansal, Varun Tolani, Saurabh Gupta, Jitendra Malik, and Claire Tomlin. Combining optimal control and learning for visual navigation in novel environments. *arXiv preprint arXiv:1903.02531*, 2019.

Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1-2):41–77, 2003.

John Canny. *The complexity of robot motion planning*. MIT press, 1988.

Luca Carlone, Jingjing Du, Miguel Kaouk Ng, Basilio Bona, and Marina Indri. Active slam and exploration with particle filters using kullback-leibler divergence. *Journal of Intelligent & Robotic Systems*, 75(2):291–311, 2014.

Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *arXiv preprint arXiv:1709.06158*, 2017.

Devendra Singh Chaplot, Emilio Parisotto, and Ruslan Salakhutdinov. Active neural localization. *ICLR*, 2018.

Tao Chen, Saurabh Gupta, and Abhinav Gupta. Learning exploration policies for navigation. *arXiv preprint arXiv:1903.01959*, 2019.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014.

Christian Dornhege and Alexander Kleiner. A frontier-void-based approach for autonomous exploration in 3d. *Advanced Robotics*, 27(6):459–468, 2013.

Kuan Fang, Alexander Toshev, Li Fei-Fei, and Silvio Savarese. Scene memory transformer for embodied agents in long-horizon tasks. In *CVPR*, 2019.

J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha. Visual simultaneous localization and mapping: a survey. *Artificial Intelligence Review*, 2015.

Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. Iqa: Visual question answering in interactive environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4089–4098, 2018.

Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. *arXiv preprint arXiv:1702.03920*, 3, 2017.

Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Joao F Henriques and Andrea Vedaldi. Mapnet: An allocentric spatial memory for mapping environments. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8476–8484, 2018.

Dirk Holz, Nicola Basilico, Francesco Amigoni, and Sven Behnke. Evaluating the efficiency of frontier-based exploration strategies. In *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*, pages 1–8. VDE, 2010.

Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. *UIST*, 2011.

Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015.

Elia Kaufmann, Mathias Gehrig, Philipp Foehn, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Beauty and the beast: Optimal methods meet learning for drone racing. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 690–696. IEEE, 2019.

Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *RA*, 1996.

Arbaaz Khan, Clark Zhang, Nikolay Atanasov, Konstantinos Karydis, Daniel D Lee, and Vijay Kumar. End-to-end navigation in unknown environments using neural networks. *arXiv preprint arXiv:1707.07385*, 2017.

S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.

Thomas Kollar and Nicholas Roy. Trajectory optimization using reinforcement learning for map exploration. *The International Journal of Robotics Research*, 27(2):175–196, 2008.

Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`, 2018.

Guillaume Lample and Devendra Singh Chaplot. Playing FPS games with deep reinforcement learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Steven M Lavalle and James J Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, 2000.

Lisa Lee, Emilio Parisotto, Devendra Singh Chaplot, Eric Xing, and Ruslan Salakhutdinov. Gated path planning networks. *arXiv preprint arXiv:1806.06408*, 2018.

Ruben Martinez-Cantin, Nando de Freitas, Eric Brochu, José Castellanos, and Arnaud Doucet. A bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot. *Autonomous Robots*, 27(2):93–103, 2009.

Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *ICLR*, 2017.

Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

Adithyavairavan Murali, Tao Chen, Kalyan Vasudev Alwala, Dhiraj Gandhi, Lerrel Pinto, Saurabh Gupta, and Abhinav Gupta. Pyrobot: An open-source robotics framework for research and benchmarking. *arXiv preprint arXiv:1906.08236*, 2019.

Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *2011 international conference on computer vision*, pages 2320–2327. IEEE, 2011.

Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *ICLR*, 2018.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Andrzej Pronobis and Patric Jensfelt. Large-scale semantic mapping and reasoning with heterogeneous modalities. In *2012 IEEE International Conference on Robotics and Automation*, pages 3515–3522. IEEE, 2012.

Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. *arXiv preprint arXiv:1803.00653*, 2018a.

Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability. *arXiv preprint arXiv:1810.02274*, 2018b.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. *arXiv preprint arXiv:1904.01201*, 2019.

Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pages 222–227, 1991.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

James A Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996.

Cyrill Stachniss, Giorgio Grisetti, and Wolfram Burgard. Information gain-based exploration using rao-blackwellized particle filters. In *Robotics: Science and Systems*, volume 2, pages 65–72, 2005.

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

Matthew R Walter, Sachithra Hemachandra, Bianca Homberg, Stefanie Tellex, and Seth Teller. Learning semantic maps from natural language descriptions. Robotics: Science and Systems, 2013.

Fei Xia, Amir R. Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson Env: real-world perception for embodied agents. In *Computer Vision and Pattern Recognition (CVPR), 2018 IEEE Conference on*. IEEE, 2018.

Kai Xu, Lintao Zheng, Zihao Yan, Guohang Yan, Eugene Zhang, Matthias Niessner, Oliver Deussen, Daniel Cohen-Or, and Hui Huang. Autonomous reconstruction of unknown indoor scenes guided by time-varying tensor fields. *ACM Transactions on Graphics (TOG)*, 36(6):202, 2017.

Brian Yamauchi. A frontier-based approach for autonomous exploration. In *cira*, volume 97, page 146, 1997.

Jingwei Zhang, Lei Tai, Joschka Boedecker, Wolfram Burgard, and Ming Liu. Neural slam: Learning to explore with external memory. *arXiv preprint arXiv:1706.09520*, 2017.

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3357–3364. IEEE, 2017.

Table 3: 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.

				Domain Generalization		Goal Generalization			
	Test Setting →	Gibson Val		MP3D Test		Hard-GEDR		Hard-Dist	
Train Task	Method	Succ	SPL	Succ	SPL	Succ	SPL	Succ	SPL
PointGoal	Random	0.027	0.021	0.010	0.010	0.000	0.000	0.000	0.000
	RL + Blind	0.625	0.421	0.136	0.087	0.052	0.020	0.008	0.006
	RL + 3LConv + GRU	0.550	0.406	0.102	0.080	0.072	0.046	0.006	0.006
	RL + Res18 + GRU	0.561	0.422	0.160	0.125	0.176	0.109	0.004	0.003
	RL + Res18 + GRU + AuxDepth	0.640	0.461	0.189	0.143	0.277	0.197	0.013	0.011
	RL + Res18 + GRU + ProjDepth	0.614	0.436	0.134	0.111	0.180	0.129	0.008	0.004
	IL + Res18 + GRU	0.823	0.725	0.365	0.318	0.682	0.558	0.359	0.310
	CMP	0.827	0.730	0.320	0.270	0.670	0.553	0.369	0.318
	ANM	**0.951**	**0.848**	**0.593**	**0.496**	**0.824**	**0.710**	0.662	**0.534**
Exploration	ANM - Task Transfer	0.950	0.846	0.588	0.490	0.821	0.703	**0.665**	0.532

A POINTGOAL EXPERIMENTS

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. 2019, 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[5]. 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. 2019 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 $6.9/6.5/7.0$m in train/val/test splits (Savva et al., 2019). 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. 2019. 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. 2018.

A.1 POINTGOAL RESULTS

In Table 3, 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 (10million frames). The proposed model achieves a success rate/SPL of 0.950/0.846 as compared to 0.827/0.730 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.

[5]`https://evalai.cloudcv.org/web/challenges/challenge-page/254`

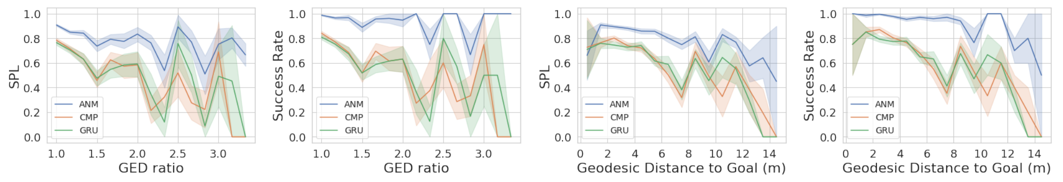


Figure 6: Performance of the proposed ANM model along with CMP and IL + Res18 + GRU (GRU) baselines with increase in geodesic distance to goal and increase in GED Ratio on the Gibson Val set.

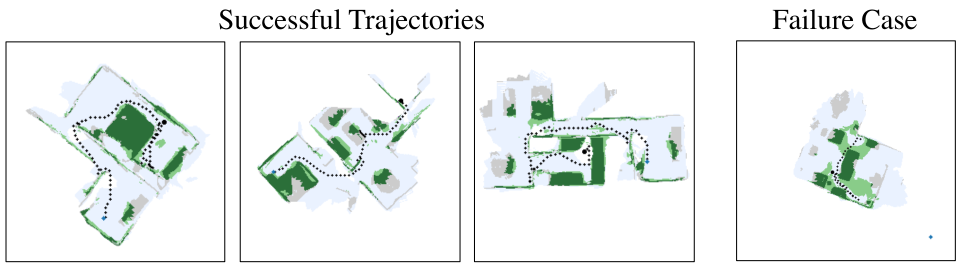
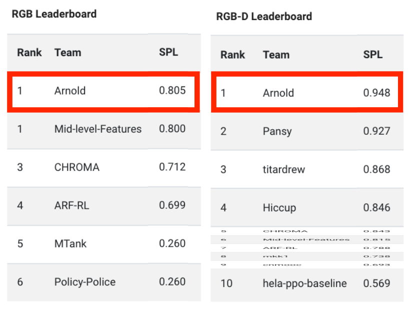


Figure 7: Figure showing sample trajectories of the proposed model along with predicted map in the PointGoal task. The starting and goal locations are shown by black squares and blue circles, respectively. Ground truth map is under-laid in grey. Map prediction is overlaid in green, with dark green denoting 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.

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. ANM reaches the performance of 0.789/0.703 SPL/Succ at only 1 million frames. These numbers indicate that ANM achieves $> 75\times$ speedup as compared to the best RL baseline.

Domain and Goal Generalization: In Table 3 (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 6. 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 9, 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.[6]



RGB Leaderboard

Rank	Team	SPL
1	Arnold	0.805
1	Mid-level-Features	0.800
3	CHROMA	0.712
4	ARF-RL	0.699
5	MTank	0.260
6	Policy-Police	0.260

RGB-D Leaderboard

Rank	Team	SPL
1	Arnold	0.948
2	Pansy	0.927
3	titardrew	0.868
4	Hiccup	0.846
5	CHROMA	0.843
6	Mid-level-Features	0.815
7	ARF-RL	0.788
8	mkk1	0.738
9	cnmooc	0.693
10	hela-ppo-baseline	0.569

Figure 5: Screenshot of CVPR 2019 Habitat Challenge Results. The proposed model was submitted under code-name ‘Arnold’.

Habitat Challenge Results. We submitted the ANM model to the CVPR 2019 Habitat Pointgoal Navigation Challenge. The results are shown in Figure 5. ANM was submitted under code-name ‘Arnold’. ANM was the winning entry for both RGB and RGB-D tracks among over 150 submissions from 16 teams, achieving an SPL of 0.805 (RGB) and 0.948 (RGB-D) on the Test Challenge set.

[6]See `https://sites.google.com/view/active-neural-mapping` for visualization videos.

B Additional visualizations

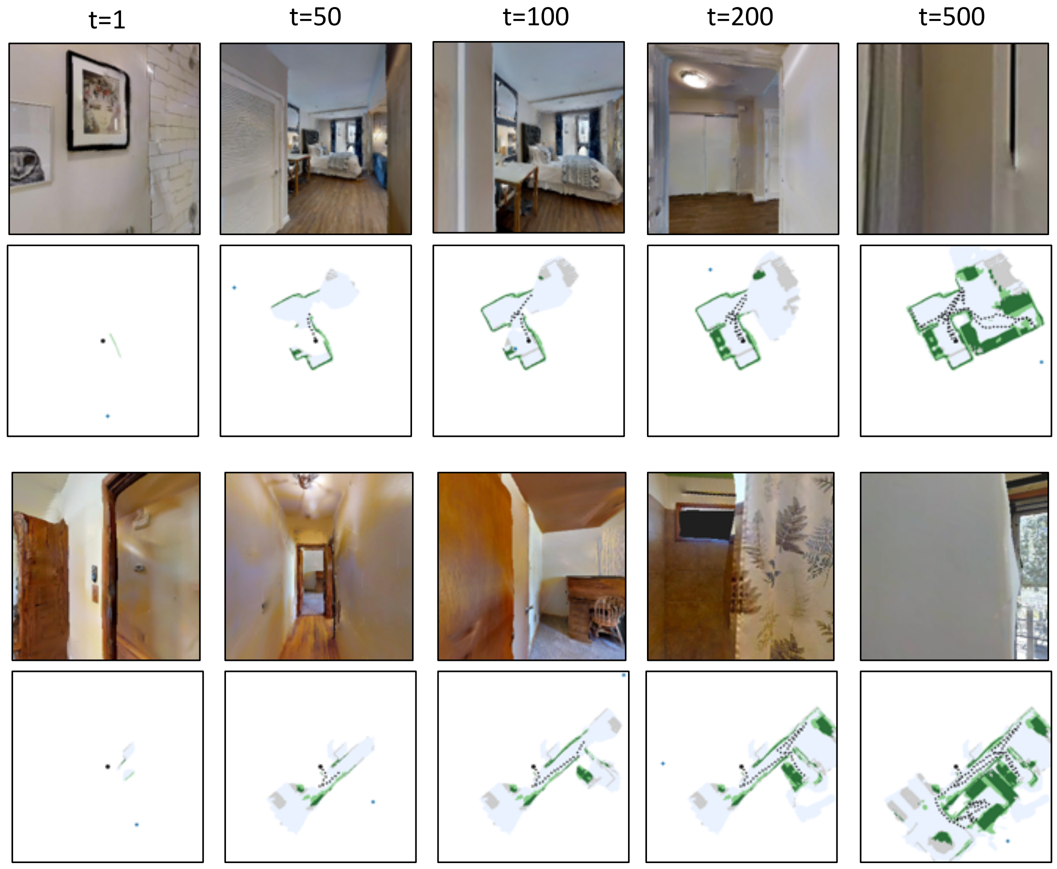


Figure 8: Exploration visualization. Figure showing sample trajectories of the proposed model along with predicted map in the Exploration task as the episode progresses. The starting location is shown in black square. Long-term goals selected by the Global policy are shown by blue circles. Ground truth 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.

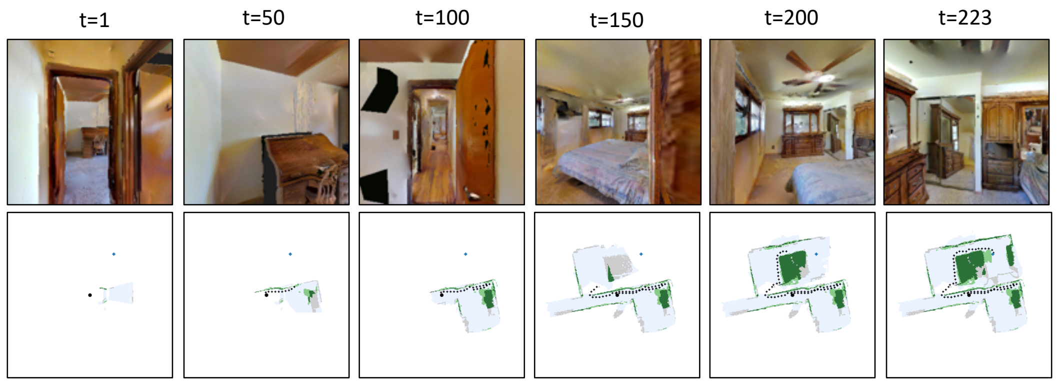


Figure 9: Pointgoal visualization. Figure showing sample trajectories of the proposed model along with predicted map in the Pointgoal task as the episode progresses. The starting and goal locations are shown by black squares and blue circles, respectively. Ground truth map is under-laid in grey. Map 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.

C Noise model implementation details

In order to implement the actuation and sensor noise models, we would like to collect data for navigational actions in the Habitat simulator. We use three default navigational actions: Forward: move forward by 25cm, Turn Right: on the spot rotation clockwise by 10 degrees, and Turn Left: on the spot rotation counter-clockwise by 10 degrees. The control commands are implemented as $u_{Forward} = (0.25, 0, 0)$, $u_{Right} : (0, 0, -10 * \pi/180)$ and $u_{Left} : (0, 0, 10 * \pi/180)$. In practice, a robot can also rotate slightly while moving forward and translate a bit while rotating on-the-spot, creating rotational actuation noise in forward action and similarly, a translation actuation noise in on-the-spot rotation actions.

We use a Locobot [7] to collect data for building the actuation and sensor noise models. We use the pyrobot API (Murali et al., 2019) along with ROS (Quigley et al., 2009) to implement the control commands and get sensor readings. In order to get an accurate agent pose, we use an Hokuyo UST-10LX Scanning Laser Rangefinder (LiDAR) which is especially very precise in our scenario as we take static readings in 2D (Kohlbrecher et al., 2011). We install the LiDAR on the Locobot by replacing the arm with the LiDAR. We note that the Hokuyo UST-10LX Scanning Laser Rangefinder is an expensive sensor. It costs \$1600 as compared to the whole Locobot costing less than \$2000 without the arm. Using expensive sensors can improve the performance of a model, however for a method to be scalable, it should ideally work with cheaper sensors too. In order demonstrate the scalability of our method, we use the LiDAR only to collect the data for building noise models and not for training or deploying navigation policies in the real-world.

For the sensor estimate, we use the Kobuki base odometry available in Locobot. We approximate the LiDAR pose estimate to be the true pose of the agent as it is orders of magnitude more accurate than the base sensor. For each action, we collect 600 datapoints from both the base sensor and the LiDAR, making a total of 3600 datapoints ($600 * 3 * 2$). We use 500 datapoints for each action to fit the actuation and sensor noise models and use the remaining 100 datapoints for validation. For each action a, the LiDAR pose estimates gives us samples of p_1 and the base sensor readings give us samples of $p'_1, i = 1, 2, \ldots, 600$. The difference between LiDAR estimates (p_1^i) and control command (Δu_a) gives us samples for the actuation noise for the action a: $\epsilon^i_{act,a} = p^i_1 - \Delta u_a$ and difference between base sensor readings and LiDAR estimates gives us the samples for the sensor noise, $\epsilon^i_{sen,a} = p^{i'}_1 - p^i_1$.

For each action a, we fit a separate Gaussian Mixture Model for the actuation noise and sensor noise using samples $\epsilon^i_{act,a}$ and $\epsilon^i_{sen,a}$ respectively, making a total of 6 models. We fit Gaussian mixture models with number of components ranging from 1 to 20 for and pick the model with highest likelihood on the validation set. Each component in these Gaussian mixture models is a multi-variate Gaussian in 3 variables, x, y and o. We implement these actuation and sensor noise models in the Habitat simulator for our experiments.

D Mapper implementation details

The Mapper (f_{Map}) takes in the current RGB observation, $s_t \in \mathbb{R}^{3\times H\times W}$, the current and last sensor reading of the agent pose $x'_{t-1:t}$ 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):

$$m_t, \hat{x}_t = f_{Map}(s_t, x'_{t-1:t}, \hat{x}_{t-1}, m_{t-1}|\theta_M, b_{t-1})$$

where θ_M denote the trainable parameters and p_{t-1} denotes internal representations of the Mapper. The Mapper can be broken down into two parts, a Projection Unit (f_{Pr}) and a Pose Estimator Unit (f_{PE},). The Projection Unit outputs a egocentric top-down 2D spatial map, $p^{ego}_t \in [0, 1]^{2\times V\times V}$ (where V is the vision range), predicting the obstacles and the explored area in the current observation: $p^{ego}_t = f_{Pr}(s_t|\theta_{Pr})$, where θ_{Pr} 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.

[7] http://locobot.org

Now, we would like to add the egocentric map prediction (p_t^{ego}) to the geocentric map from the previous time step (m_{t-1}). In order to transform the egocentric map to the geocentric frame, we need the pose of the agent in the geocentric frame. The sensor reading x'_t is typically noisy. Thus, we have a Pose Estimator to correct the sensor reading and give an accurate estimate of the agent's geocentric pose.

In order to estimate the pose of the agent, we first calculate the relative pose change (dx) from the last time step using the sensor readings at the current and last time step (x'_{t-1}, x'_t). Then we use a Spatial Transformation (Jaderberg et al., 2015) on the egocentric map prediction at the last frame (p_{t-1}^{ego}) based on the relative pose change (dx), $p'_{t-1} = f_{ST}(p_{t-1}^{ego}|dx)$. Note that the parameters of this Spatial Transformation are not learnt, but calculated using the pose change (dx). This transforms the projection at the last step to the current egocentric frame of reference. If the sensor was accurate, p'_{t-1} would highly overlap with p_t^{ego}. The Pose Estimator Unit takes in p'_{t-1} and p_t^{ego} as input and predicts the relative pose change: $\hat{dx}_t = f_{PE}(p'_{t-1}, p_t^{ego}|\theta_{PE})$ The intuition is that by looking at the egocentric predictions of the last two frames, the pose estimator can learn to predict the small translation and/or rotation that would align them better. The predicted relative pose change is then added to the last pose estimate to get the final pose estimate $\hat{x_t} = \hat{x}_{t-1} + \hat{dx}_t$.

Finally, the egocentric spatial map prediction is transformed to the geocentric frame using the current pose prediction of the agent ($\hat{x_t}$) using another Spatial Transformation and aggregated with the previous spatial map (m_{t-1}) using Channel-wise Pooling operation: $m_t = m_{t-1} + f_{ST}(p_t|\hat{x_t})$.

Combing all the functions and transformations:

$$\begin{aligned}
m_t &= f_{Map}(s_t, x'_{t-1:t}, m_{t-1}|\theta_M, b_{t-1}) \\
&= m_{t-1} + f_{ST}(p_t|x'_t + f_{PE}(f_{ST}(p_{t-1}^{ego}|x_{t-1:t}), f_{Pr}(s_t|\theta_{Pr})|\theta_{PE})) \\
&\quad \text{where } \theta_{Pr}, \theta_{PE} \in \theta_M, \quad \text{and} \quad p_{t-1}^{ego} \in b_{t-1}
\end{aligned}$$

E Architecture details

We use PyTorch Paszke et al. (2017) for implementing and training our model. 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 Pose Estimator consists of 3 convolutional layers followed by 2 fully connected layers. 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 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. Our PPO Schulman et al. (2017) implementation of Global and Local policy is based on Kostrikov 2018. In addition to the RGB observation, the Local policy receives relative distance and angle to short-term goal, current timestep and last action as input. We bin the relative distance (bin size increasing with distance), relative angle (5 degree bins) and current timestep (30 time step bins) before passing them through embedding layers. This kind of discretization is used previously for RL policies Lample and Chaplot (2017) and it improved the sample efficiency as compared to passing the continuous values as input directly. For fair comparison, we use the same discretization for all the baselines as well.

F Hyperparameter details

We train all the components with 72 parallel threads, with each thread using one of the 72 scenes in the Gibson training set. This leads to a batch size of 72 for training the Mapper. The Global policy samples a new goal every 25 timesteps. We use Proximal Policy Optimization Schulman et al. (2017) for training the Global and Local policies with 72 parallel threads and a horizon length of 25 steps for the Local policy and 20 steps for the Global policy (20 steps for Global policy is equivalent to 500 low-level timesteps as Global policy samples a new goal after every 25 timesteps). We use Adam optimizer with a learning rate of 0.0001 for training both the units in the Mapper and Adam with a learning rate of 0.00025 for training the Global and Local policies. We use a discount factor of

$\gamma = 0.99$, entropy coefficient of 0.001, value loss coefficient of 0.5 for training both the Global and Local policies.

Input frame size is 128×128, the vision range for mapper is $V = 64$, i.e. $3.2m$ (each cell is $5cm$ in length). Since there are no parameters dependent on the map size, it can be adaptive. We train with a map size of $M = 960$ (equivalent to $48m$). A map of size $48m \times 48m$ is large enough for all scenes in the Gibson val set. We use an adaptive map size for Pointgoal evaluation such that goal lies within central 50% of the map to handle even larger maps in the unseen test set. For the exploration task, we train and test with a constant $M = 960$. For the Global policy in the Exploration task, the size of the Global Policy input is $G = 240$.

G Additional Results

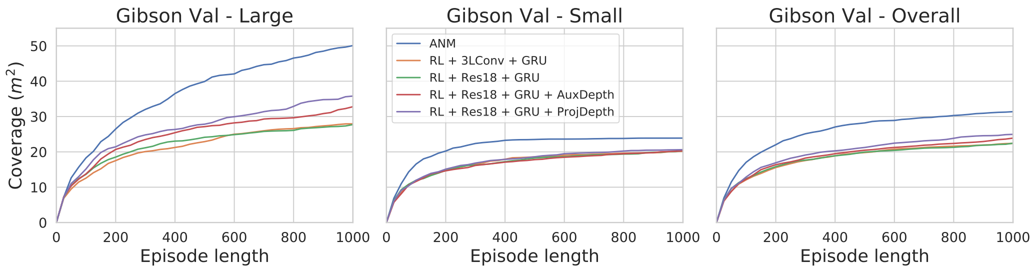


Figure 10: Plot showing the % Coverage as the episode progresses for ANM and the baselines on the large and small scenes in the Gibson Val set as well as the overall Gibson Val set.