2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Scaling Local Control to Large-Scale Topological Navigation

Xiangyun Meng, Nathan Ratliff, Yu Xiang and Dieter Fox

***Abstract*— Visual topological navigation has been revitalized recently thanks to the advancement of deep learning that substantially improves robot perception. However, the scalability and reliability issue remain challenging due to the complexity and ambiguity of real world images and mechanical constraints of real robots. We present an intuitive approach to show that by accurately measuring the capability of a local controller, large-scale visual topological navigation can be achieved while being scalable and robust. Our approach achieves state-of-the-art results in trajectory following and planning in large-scale environments. It also generalizes well to real robots and new environments without retraining or finetuning.**

I. Introduction

There has been an emergence of cognitive approaches [1], [2], [3], [4], [5] towards navigation thanks to the advancement of deep learning that substantially improves robot perception. Compared to the traditional mapping, localization and planning approach (SLAM) [6], [7] that builds a metric map, cognitive navigation uses a topological map. This eliminates the need of meticulously reconstructing an environment which requires expensive or bulky hardware such as a laser scanner or a high-resolution camera. Moreover, the fact that humans are able to navigate effortlessly in large-scale environments without a metric map is intriguing. By adding this cognitive spatial reasoning capability to robots, we could potentially lower the hardware cost (i.e., using low-resolution cameras), make them work more robustly in dynamic environments and bring insights to more complex tasks such as visual manipulation.

While cognitive navigation has drawn significant attention recently, the problem remains challenging because i) it does not scale well to the size of experiences ii) it is fragile due to actuation noise and dynamic obstacles and iii) it lacks probabilistic interpretation, making it difficult to plan with uncertainty. These problems are exacerbated when using a RGB camera in indoor environments, where partial observability makes it difficult to control a robot to follow a single path [3], [8].

In this paper, we present a simple and intuitive solution for topological navigation. We show that by accurately measuring the capability of a local controller, robust visual topological navigation can be achieved with sparse experiences (Fig.1). In our approach, we do not assume the availability of a global coordinate system or robot poses, nor do we assume noise-free actuation or static environment. This minimalistic representation only has two components: a local controller and a reachability estimator. The controller is responsible for local reactive navigation, whereas the reachability estimator measures the *capability* of the controller for landmark selection and long-term probabilistic planning. To achieve this, we leverage the Riemannian Motion Policy (RMP) framework [9] for robust reactive control and deep learning for learning the capability of the controller from data. We show that with both components working in synergy, a robot can i) navigate robustly with the presence of nonholonomic constraints, actuation noise and obstacles; ii) build a compact spatial memory through adaptive experience sparsification and iii) plan in the topological space probabilistically, allowing robot to generalize to new navigation tasks.

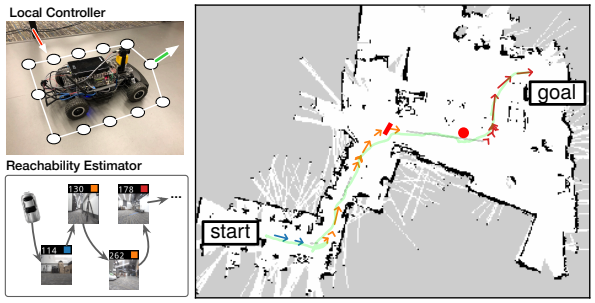


Fig. 1: Overview of our method. The local controller drives the vehicle towards a given target image, and the reachability estimator plans a path by combining multiple experiences (colored arrows on the map) to provide the controller a sequence of target observations (bottom left) to follow. The vehicle is able to navigate robustly in the real environment (right) while avoiding unseen obstacles (red rectangle and circle). The model is trained entirely in simulation.

We evaluate our approach in the Gibson simulation environment [10] and on a real RC car. Our test environments contain a diverse set of real-world indoor scenes with presence of strong symmetry and tight spaces. We show that our approach generalizes well to these unseen environments and surprisingly well to real robots without finetuning. Scalability-wise, our spatial memory grows only when new experiences are unseen, making the system space-efficient and compute-efficient.

II. Related Work

Cognitive spatial reasoning has been extensively studied both in neuroscience [11], [12], [13], and robotics [14], [15], [16]. The Spatial Semantic Hierarchy [16] divides the cognitive mapping process into four levels: control, causal, topological and metric. In our method, the local controller operates on the control level, whereas the reachability estimator reasons about causal and topological relationship between observations. We omit metric-level reasoning since we are not concerned about building a metric map.

Experience-driven navigation constructs a topological map for localization and mapping [15], [17], [18], [19], [4]. Unlike SLAM that assumes a static environment, the experience

Xiangyun Meng and Dieter Fox are with the Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, WA 98195, USA {xiangyun, fox}@cs.washington.edu
Nathan Ratliff, Yu Xiang and Dieter Fox are with NVIDIA, Seattle, WA 98105, USA { nratliff, yux, dieterf}@nvidia.com

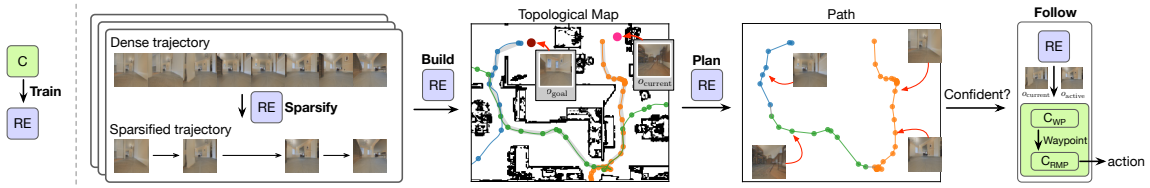


Fig. 2: System overview. Given a controller $\mathbb{C}$, we train a reachability estimator $\mathbb{RE}$. $\mathbb{RE}$ is used for sparsifying incoming trajectories, building a compact topological map and planning a path. $\mathbb{C}$ and $\mathbb{RE}$ work in synergy to robustly follow the planned path.

graph can also be used for dealing with long-term appearance changes [20]. This line of works mostly focus on appearance-based localization and ignore the control aspect of navigation, and assume that a robot can always follow experiences robustly. This does not usually hold in unstructured indoor environments, where it is crucial to design a good controller while considering its capability.

Semi-Parametric Topological Memory (SPTM) [1], [21] is a recent work that adopts deep learning into topological navigation. Similar to SPTM, we build a topological map through past experiences. Unlike SPTM that uses image similarity as a proxy for reachability, we measure the reachability of a controller directly. This significantly improves robustness and opens opportunities for constructing sparse maps.

There have been recent works studying visual trajectory following that handles obstacles [8], [22], actuation noise [3], or with self-supervision [23]. Our approach differs from them in that our trajectory follower extends seamlessly to probabilistic planning. Our method also handles obstacles and actuation noise well, thanks to the RMP controller that models local geometry and vehicle dynamics.

Recent works on cognitive planning [24], [25] show that a neural planner can be learned from data. However, assumptions such as groundtruth camera poses are available with perfect self-localization are unrealistic. The use of grid map also limits its flexibility. Another line of research uses reinforcement learning to learn a latent map [26], [27], but it is data-inefficient and cannot be easily applied to real robots. In contrast, our planner is general and can adapt to new environments quickly. It bears a resemblance to feedback motion planning system such as LQR-Trees [28], where planning is performed on the topological map connecting reachable state spaces with visual feedback control.

III. Method

A. Overview

We consider the goal-directed navigation problem: a robot is asked to navigate to a goal G given an observation o_G taken at G. Robot does not have a map of the environment, but we assume it has collected a set of trajectories (e.g., via self-exploration or following language instructions) as its experiences. Each trajectory is a dense sequence of observations $o_1, o_2, ..., o_N$ recorded by its on-board camera. Using its experiences, robot decides the next action to take in order to reach G. The action space is continuous (e.g., velocity and steering angle) and robot could be affected by actuation noise and unseen obstacles.

We approach this problem from a cognitive perspective. Robot first builds a topological map from its experiences. The map is a directed graph, with vertices as observations and edges encoding traversability. Then, given its current observation o_{current} and goal o_G, robot searches for a path on the graph and follows that path to reach G. Our setup is similar to that of SPTM [1]. The difference is that we design our system to make it generalize to real robots and scale to real environments.

For such a navigation system to work, we first need a target-conditioned **Local Controller** $\mathbb{C}$. $\mathbb{C}$ takes current observation and a target observation, and outputs an action $a = \mathbb{C}(o_{\text{current}}, o_{\text{target}})$ to drive robot towards the target. The action is executed for a small time step to get an updated o_{current} and the process is repeated until o_{current} matches o_{target}. Given a path (a sequence of observations) computed by a planner, robot uses $\mathbb{C}$ to follow the path progressively to reach its final destination.

In practice, robot's experience pool can be large and grow indefinitely, thus the key issue is to build a sparse and scalable representation of an environment given dense, unstructured trajectories. Clearly, adjacent observations in a trajectory is highly correlated and it would be wasteful to keep every observation. One ad-hoc approach to sparsify a trajectory is to take every nth observation. However, this assumes that target n steps away is always reachable, which is not necessarily true. For example, without occlusion, an observation far away can be confidently reached (e.g., in a straight hallway), whereas an observation nearby may be hidden (thus not reachable) if it is blocked by obstacles. Moreover, motion constraints, sensor field of view, motor noise, etc. can all affect the reachability of a target.

Our intuition is that the sparsification of a trajectory should adapt to the capability of the controller. We propose learning a **Reachability Estimator** $\mathbb{RE}$ that predicts the probability of $\mathbb{C}$ successfully reaching a target: $\mathbb{RE}(o_{\text{current}}, o_{\text{target}}) = P(\text{reach}|o_{\text{current}}, o_{\text{target}}, \mathbb{C})$. We use $\mathbb{RE}$ as a probabilistic metric throughout the system, illustrated by Fig. 2. Given a controller $\mathbb{C}$, we train a corresponding $\mathbb{RE}$. The incoming trajectories are first sparsified by $\mathbb{RE}$ and then interlinked to form a compact topological map. Given o_{current} and o_G, we leverage $\mathbb{RE}$ to plan a probabilistic path and use $\mathbb{C}$ and $\mathbb{RE}$ in synergy to follow the planned path robustly.

B. Designing a Robust Local Controller

Real-world robots are subject to disturbances such as motor noise and moving obstacles, which can cause a robot to deviate from planned path and fail. Hence our first objective is to design a sufficiently robust local controller. Contrary to directly predicting low-level controls, we split our controller into two stages: high-level waypoint prediction and low-level

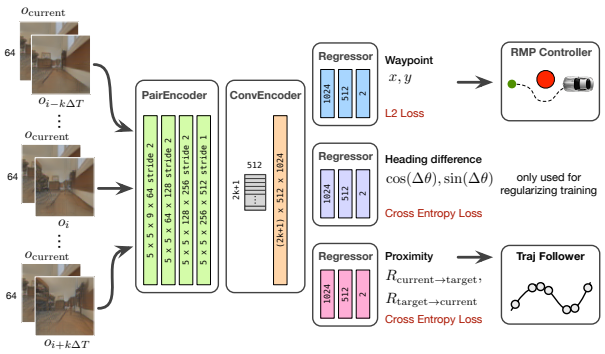


Fig. 3: Architecture of $\mathbb{C}_{\text{WP}}$. The architecture of $\mathbb{RE}$ is similar, except that it regresses to a single probability and is supervised with cross-entropy loss.

	Control(k=0)	Ours(k=0)	Ours(k=2)	Ours(k=5)
Success%	46%	88%	91%	95%

TABLE I: Success rate for each controller.

reactive control. The high-level controller $\mathbb{C}_{\text{WP}}$ predicts a waypoint x, y (in robot's local coordinate system) for the low-level controller. The waypoint needs not be precise, but only serves as a hint for the low-level controller to make progress. Hence $\mathbb{C}_{\text{WP}}$ is agnostic to robot dynamics (e.g., can be trained with A* waypoints as supervision) and absorbs the effects of actuation noise. For the low-level controller, we adopt the RMP representation [29] as a principled way for obstacle avoidance and vehicle control. Hence we have $\mathbb{C}(o_{\text{current}}, o_{\text{target}}) = \mathbb{C}_{\text{RMP}}(\mathbb{C}_{\text{WP}}(o_{\text{current}}, o_{\text{target}}))$. Note that this allows the same $\mathbb{C}_{\text{WP}}$ to be applied to different robots by replacing the low-level controller.

Fig. 3 illustrates the design of $\mathbb{C}_{\text{WP}}$. The robot state is represented by its current observation o_{current}. Denote ith observation in a trajectory as o_i. We represent the corresponding o_{target} at o_i as a sequence of neighbor observations centered at o_i:

$$o_{i-k\Delta T}, o_{i-(k-1)\Delta T}, ..., o_i, ..., o_{i+(k-1)\Delta T}, o_{i+k\Delta T},$$

where k controls context length and ΔT (set to 3) is the gap between two observations. The past frames expand the field of view of o_i which helps controller to do visual closed-loop control. The future frames encode intention at o_i, allowing a controller to adjust its waypoint in advance in order to follow subsequent targets smoothly and reliably.

Technically, we extract a feature vector by feeding stacked $[o_{\text{current}}, o_{i-k\Delta T}, o_{\text{current}} - o_{i-k\Delta T}]$ into a sequence of convolutions, followed by combining the $2k+1$ feature vectors through one convolution and multiple fully-connected layers to predict a waypoint x, y. We find this design works much better than featurizing each image or stacking all images together. Additionally, the network predicts the heading difference between o_{current} and o_i to help the network anchor the target image in the sequence. Finally, in order to reason about proximity to a target (Sec.III-H), $\mathbb{C}_{\text{WP}}$ predicts mutual image overlap. Image overlap is a ratio that represents the percentage of content in one image that is visible in another image. Hence mutual image overlap is a pair of ratios $(R_{\text{current}\to\text{target}}, R_{\text{target}\to\text{current}})$.

We train $\mathbb{C}_{\text{WP}}$ in a supervised fashion (Sec.IV). To evaluate our design, we randomly sample o_{current} from a trajectory and o_{target} being $-1.0m$ behind to $3.0m$ ahead of o_{current} in 4 unseen environments, and run each controller to see if robot reaches the target. Table I compares the success rate of each controller. Directly predicting low-level controls (forward acceleration and steering velocity) results in much lower success rate than our two-stage design. Compared with directly mapping images to low-level actions, we find it more robust to map images to higher-level abstractions such as waypoints, and then map waypoints to low-level controls using a representation (e.g., RMP) that explicitly models environmental geometry and robot dynamics.

C. Learning the Reachability Estimator

Table I suggests that controller design and parametrization can heavily affect target reachability. Unlike [1] that uses image similarity as a proxy, we learn reachability by explicitly predicting the execution outcome of $\mathbb{C}$. During training, o_{current} and o_{target} are randomly sampled from demonstration trajectories (Sec. IV) and $\mathbb{C}$ is used to drive the robot from o_{current} to o_{target} to get a binary outcome. The criteria for success is that robot reaches the target within time limit defined as $t_{\max} = \text{A*}(o_{\text{current}}, o_{\text{target}})/v_{\min}$, where $\text{A*}(\cdot,\cdot)$ computes the A* path length and $v_{\min}$ is the minimum velocity. Hence $\mathbb{RE}$ measures the probability of $\mathbb{C}$ reaching the target *efficiently*, which is independent of the temporal and physical distance between o_{current} and o_{target}. This idea has an interesting connection to feedback motion planning systems [28], as $\mathbb{RE}$ can be seen as estimating visual funnels that are locally stable.

The design of $\mathbb{RE}$ is almost identical to $\mathbb{C}$, except that it predicts a single probability and is trained with a binary classification loss.

D. Sparsifying a Trajectory

For any observation o_i in a dense trajectory, if $\mathbb{RE}(o_i, o_{i+1}), ..., \mathbb{RE}(o_i, o_{i+k+1})$ are sufficiently high, we could confidently discard $o_{i+1}, ..., o_{i+k}$ because $\mathbb{C}$ does not need them to reach o_{i+k+1}. Hence a greedy approach to choose the next anchor is

$$\max\ j$$
$$\text{s.t. } \mathbb{RE}(o_i, o_k) > p_{\text{sparsify}}, \forall k, i < k \leq j$$

where i is previous anchor's position and p_{sparsify} is the probability threshold that controls sparsity. Hence a dense trajectory is converted to a sequence of contextified anchor observations $\hat{o}_1, ..., \hat{o}_m$. One may argue that contextification reduces the effective sparsification ratio. Since the time and space complexity is a function of the number of anchors, in practice it significantly saves computation during planning and following a trajectory, allowing our system to run on a robot in real time.

E. Building a Compact Probabilistic Topological Map

Our topological map is a weighted directed graph (Fig. 4a). Vertices are anchor observations and edge weight from $\hat{o}_i$ to $\hat{o}_j$ is $-\log \mathbb{RE}(\hat{o}_i, \hat{o}_j)$. Construction is incremental: for an

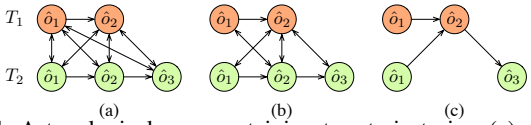


Fig. 4: A topological map containing two trajectories. (a) densely connected graph. (b) after pruning low-probability edges. (c) after reusing nodes.

incoming trajectory, we create pairwise edges between every vertex in the graph and every anchor in the new trajectory.

Compared to a graph constructed with dense observations, a graph built from sparsified observations has less than 1/10 of the vertices and 1/100 of the edges. To further improve scalability, we propose the following two optimizations to make the graph grow sublinearly to the number of raw observations, and eventually the size of the graph converges:

Edge pruning. Low-probability edges with $\mathbb{RE}(\hat{o_i}, \hat{o_j}) < p_{\text{edge}}$ are discarded since they contribute little to successful navigation (Fig. 4b).

Vertex reuse. It is common for two trajectories to be partially overlapped and storing this overlapping part repeatedly is unnecessary. Hence when adding anchor $\hat{o}_i$ into a graph, we check if there exists a vertex $\hat{o}$ such that $\mathbb{RE}(\hat{o}_{i-1}, \hat{o}) > p_{\text{reuse}}$ and $\mathbb{RE}(\hat{o}, \hat{o}_{i+1}) > p_{\text{reuse}}$. If the condition holds, we discard $\hat{o}_i$ and add edges $\hat{o}_{i-1} \rightarrow \hat{o}$ and $\hat{o} \rightarrow \hat{o}_{i+1}$, as illustrated in Fig. 4c.

The graph will converge because for any static environment of finite size, there is a maximum density of anchors. Any additional anchor will pass the vertex reuse check and be discarded. Practically however, an environment may change over time. The solution is to timestamp every observation and discard outdated observations using $\mathbb{RE}$. We leave the handling of long-term appearance change as future work.

F. Planning

We add an edge (weighted by its negative log probability) from o_{current} to every vertex in the graph, and from every vertex in the graph to o_G. The weighted Dijkstra algorithm computes the path with the lowest negative log probability (i.e., the path that robot is most confident). Robot then decides whether the probability is high enough and may run the trajectory follower proposed in Section III-H.

G. Mitigating Perceptual Aliasing

Practically, o_{current} may correspond to different locations of similar appearances. Traditional approaches usually formulate this as a POMDP problem [6] and try to resolve the ambiguity by maintaining beliefs over states. This requires having a unique state (e.g., global pose) associated with each observation which is difficult to implement since we do not have any metric information.

We use two techniques to resolve ambiguity. The first is to match a sequence of anchors during search and graph construction. In practice the probability of two segments having similar appearances is much lower than two single observations. Additionally we let robot re-plan a new path if it detects discrepancy (entering *Dead reckoning* state for too long) while following the previous path. The intuition is that the location where robot detects the discrepancy is likely distinct. See Sec. IV-C.4 for an example. In the worst case where such distinctive anchor is absent, robot might follow a cycle of anchors without making progress. The solution is to count how many times the robot has visited an anchor (i.e., by collecting statistics from *last visited anchor*). Cyclic behavior can be detected so that the robot can break the loop by biasing its choice in future planning. We leave the handling of this extreme case as future work.

H. Following a Trajectory

Our trajectory follower constantly updates and tracks an active anchor to make progress, while performing dead reckoning to counter local disturbances. Specifically, given a sequence of anchor observations $\hat{o}_1, \hat{o}_2, ..., \hat{o}_m$, the trajectory follower acts as a state machine:

Search: robot searches for the best anchor: $\hat{o}^* = \text{argmax}_{o \in \{\hat{o}_1, ..., \hat{o}_m\}} \mathbb{RE}(o_{\text{current}}, o)$. If $\mathbb{RE}(o_{\text{current}}, \hat{o}^*) > p_{\text{search}}$, it sets $\hat{o}^*$ as current active anchor $\hat{o}_{\text{active}}$ and enters *Follow* state, otherwise it gives up and stops.

Follow: robot computes the next waypoint $x, y = \mathbb{C}_{\text{WP}}(o_{\text{current}}, \hat{o}_{\text{active}})$ and uses it to drive $\mathbb{C}_{\text{RMP}}$. Meanwhile it tracks and updates the following two values:

- *last visited anchor*. Robot uses the predicted mutual image overlap to measure the proximity between o_{current} and anchors close to $\hat{o}_{\text{active}}$. The closest anchor is set as $o_{\text{lastvisited}}$. This is a form of approximate localization.
- *active anchor*. If $\mathbb{RE}(o_{\text{current}}, \hat{o}_{\text{active+1}}) > p_{\text{follow}}$ and is within proximity, it advances $\hat{o}_{\text{active}}$ to $\hat{o}_{\text{active + 1}}$, otherwise $\hat{o}_{\text{active}} = o_{\text{lastvisited + 1}}$. The intuition is to choose an $\hat{o}_{\text{active}}$ that is neither too close nor too far away.

Normally robot stays in *Follow* state. But if moving obstacles or actuation noise cause $\mathbb{RE}(o_{\text{current}}, \hat{o}_{\text{active}}) < p_{\text{follow}}$, it enters *Dead reckoning* state.

Dead reckoning: robot tracks the last waypoint computed in the *Follow* state and uses the waypoint to drive $\mathbb{C}_{\text{RMP}}$. The assumption is that disturbances are transient which the robot could escape by following the last successfully computed waypoint. Waypoint tracking can be done by an odometer and needs not be very accurate. While in this state, robot keeps checking if $\mathbb{RE}(o_{\text{current}}, \hat{o}_{\text{active}}) > p_{\text{follow}}$ and returns to *Follow* state if possible.

IV. Experiments

We trained $\mathbb{C}_{\text{WP}}$, $\mathbb{RE}$ and all baselines in 12 Gibson environments. 100k training trajectories were generated by running an A* planner (used to provide waypoints) with a laser RMP controller similar to [29]. Simulation step size is 0.1. We use the laser RMP controller as $\mathbb{C}_{\text{RMP}}$ mostly for efficiency, but in practice an image-based RMP controller can also be used [29]. $\mathbb{C}_{\text{WP}}$ was trained by randomly sampling two images on the same trajectory with certain visual overlap, with the A* waypoint as supervision. After $\mathbb{C}_{\text{WP}}$ was trained, we trained $\mathbb{RE}$ by sampling two images that either belong to the same trajectory (prob 0.6) or different trajectories (prob 0.4), and ran a rollout with $\mathbb{C}$ to get a binary outcome. Image size is 64×64 with 120° horizontal field of view. We augmented the dataset by jittering robot's

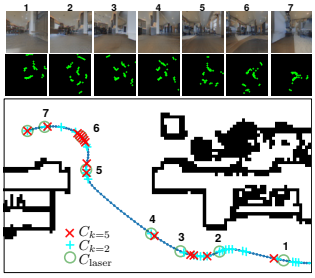


Fig. 5: Trajectory sparsification. Blue dots: dense observations. Images correspond to numbered locations. 1D laser scans are visualized from the top view.

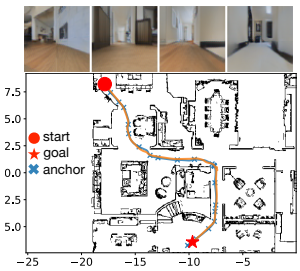


Fig. 6: Following a 23m long trajectory. Blue trace: groundtruth trajectory. Orange trace is generated by following the anchor points.

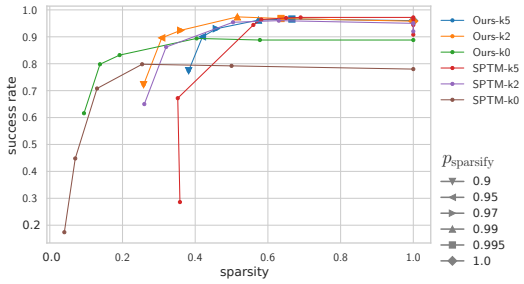
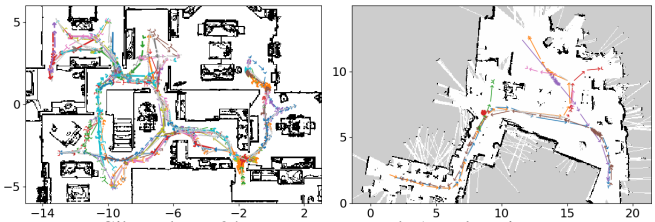


Fig. 7: Trajectory following cover rate in 5 test environments. Number after k indicates the context length. Data points for Ours-k5 and Ours-k2 are marked with p_{sparsify}.



a Gibson house24 b A real environment

Fig. 8: Example topological maps built from sparsified trajectories. Each trajectory is assigned a different color.

starting location and orientation to improve generalization. About 1.5M samples were used to train $\mathbb{C}_{\text{WP}}$ and $\mathbb{RE}$. Our training setup models a real vehicle similar to [30], so that the same model can be used for real experiments.

We present quantitative results in 5 unseen Gibson environments with diverse appearances. Our baseline is based on SPTM. Since SPTM is designed for small synthetic mazes with discrete action space, its original version would perform poorly in our setting. For a fair comparison, we let SPTM use the same controller and trajectory following logic as ours. The main differences between SPTM and ours are thus: i) how reachability is learned and ii) how graph is constructed and used. Our ablation study will thus be in the form of evaluating trajectory following and planning performance in the following sections.

A. Trajectory Sparsification

Fig. 5 compares sparsification results of three controllers. The two visual controllers $\mathbb{C}_{k=2}$, $\mathbb{C}_{k=5}$ differ in their context length. To show that our model is general, we also trained a laser-based controller $\mathbb{C}_{\text{laser}}$ by modifying the input layer in Fig. 3 to take 64-point 240° 1D depth as input.

Fig. 5 shows placement of anchors with $p_{\text{sparsify}} = 0.99$. Comparing with $\mathbb{C}_{k=5}$, $\mathbb{C}_{k=2}$ requires denser anchors. Since $\mathbb{C}_{k=2}$ uses a shorter context, it is more "local" and has to keep more anchors to follow a path robustly. Nonetheless, anchors are more densely distributed in tight spaces and corners for both controllers, indicating that our sparsification strategy adapts well to environmental geometry. Interestingly, $\mathbb{C}_{\text{laser}}$ shows a more uniform distribution pattern. Since laser scans have a much wider field of view and measures geometry directly, it is not heavily affected by tight spaces and large viewpoint change.

B. Trajectory Following

We randomly generated 500 trajectories in the test environments (Fig. 6) with an average length of 15 m. When following a trajectory, we stop the robot when it diverges from the path or collides with obstacles. We report the cover rate, the percentage of total length of trajectories successfully followed by robot. For our trajectory followers, $p_{\text{search}} = p_{\text{follow}} = 0.92$.

Sparsity is the average ratio of number of images in a sparsified trajectory to the number of images in the corresponding dense trajectory. To change sparsity, we vary p_{sparsify} for our models. For SPTM we select every nth frame and vary n. Fig. 7 plots cover rates for varying sparsity conditions. Controllers with contexts (*-k2, *-k5) achieve higher than 95% cover rate, better than controllers without context (at most 90%). This indicates that having contextual frames can improve robustness. But since contextual frames are used, more observations have to be kept so storage-wise it is not as efficient as (*-k0).

SPTM performs comparably to ours when using a strong controller (*-k5), but for all controllers it starts to degrade before ours as sparsity lowers. Due to its fixed-interval subsampling, it does not adapt to controllers' capability well, as can be seen by the increasing gap between ours and the SPTM counterparts when less contextual frames are used (*-k2, *-k0).

We also evaluated performance under noisy actuation by multiplying a random scaling factor $s \sim \mathcal{N}(1.0, 0.33)$ to the control output. No noticeable difference was found. This is expected because the local controller runs at a high frequency (10 Hz) and uses visual feedback for closed-loop control.

C. Planning

1) Navigation between Places: We built one topological map for each environment (Fig. 8a). A map is constructed from 90 trajectories connecting 10 locations in a pairwise fashion. The locations are selected to make the trajectories cover most of the reachable area.

Robot starts at one of the locations (with jittered position and orientation) and is given an goal image taken at one of the other 9 destinations. Robot has no prior knowledge of its initial location. We re-implemented SPTM's planner and uses the best trajectory follower SPTM-k5 (SecIV-B) to make it a competitive baseline. We set the sub-sampling ratio to 20 and $\Delta T_l = 1$ to prevent the graph from getting too large.

	space8	house24	house29	house75	house79
Area	$460m^2$	$207m^2$	$270m^2$	$403m^2$	$205m^2$
Images	30,342	31,167	28,679	39,788	33,617
SPTM	1,648/3,201	1,688/3,668	1,560/3,960	2,116/4,115	1,808/4,756
	48.1%	40.2%	45.6%	51.3%	47.2%
Ours	974/1,482	900/1,348	901/1,467	1,454/2,275	909/1,524
	86.9%	94.3%	91.2%	84.6%	95.7%

TABLE II: Planning success rate, with #vertices/#edges shown above. Success rate is the outcome of 1,000 navigation trials.

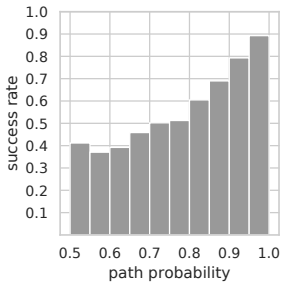


Fig. 9: Comparing path probability to empirical success rate. Each bar is the average success rate of paths whose probabilities fall into that range.

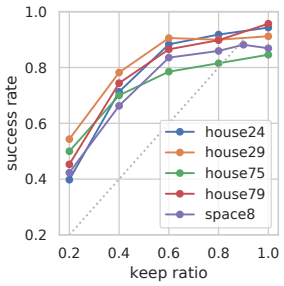


Fig. 10: Success rate with pruned graphs. Dotted line shows no-generalization for reference.

We performed a hyperparameter search to set $s_{\text{shortcut}} = 0.99$. For our method, we set $p_{\text{reuse}} = p_{\text{edge}} = 0.99$.

Table II presents the success rate for each environment compared to SPTM. Our method outperforms SPTM with much sparser maps. Graphs built by SPTM have unweighted edges and do not reuse vertices. SPTM also does explicit localization which sometimes causes planning failure. This results in worse scalability and reliability compared with our approach. Note that the slightly lower success rates in *space8* and *house75* are mostly caused by strong symmetry and rendering artefacts.

2) Comparing Trajectory Probabilities to Empirical Success Rate: To show that path probability is a reasonable indicator of empirical outcome, we let a robot start at a random location (anywhere in a map), plan a path to one of the 10 destinations, and follow the path. 1,000 trajectories were collected in each environment. Fig. 10 shows that path probability strongly corresponds to empirical success rate. This allows a robot to assess the risk before executing a plan, and ask for help if necessary. Note that SPTM does not provide any uncertainty measure.

3) Generalizing to New Navigation Tasks: To test the generalizability of our planner, we randomly pruned the graphs to contain only a subset of the trajectories, and repeated the experiment in IV-C.1. Fig. 10 shows that with only 60% of the trajectories, robot already performs close to its peak success rate. In other words, robot is able to combine existing trajectories to solve novel navigation tasks. Fig. 11 shows an example.

4) Resolving Ambiguity: Fig. 12 illustrates how perceptual aliasing is resolved in environments with strong symmetry. Robot initially starts at an ambiguous location (marked "1") and plans a wrong path (red path). While following this path, robot detects the discrepancy at "2" by realizing what is expected to be an office room is actually a hallway. As a result, it plans a new path (green) whereby it successfully reaches the goal.

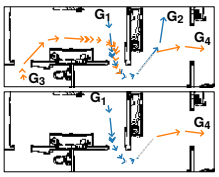


Fig. 11: Plans a path from G_1 to G_4 by combining $G_1 \rightarrow G_2$ and $G_3 \rightarrow G_4$

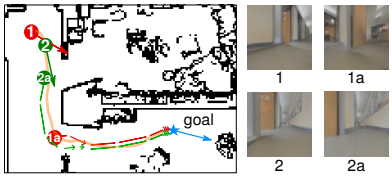


Fig. 12: Online planning. Arrows indicate headings. 1a and 2a are the next anchor observations for the two paths respectively.

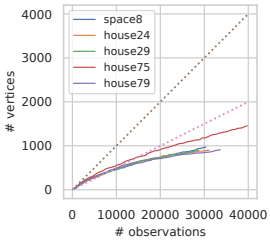


Fig. 13: Dotted lines show 10x and 20x temporal subsampling for reference.

5) Scalability: Fig. 13 shows that map sizes grow sublinearly to the number of raw observations, making our approach scalable to large environments. It also shows that map size is adaptive to an environment. Since house75 has more complex geometry and exhibits more rendering artefacts, denser samples are kept to stay conservative.

D. Testing in a Real Environment

Our model trained in simulation generalizes well to real images without finetuning. To map a real environment, we manually drove the RC car to collect 7 trajectories, totalling 3,200 images. The final map contains 206 vertices and 215 edges (Fig.8b). The car is able to plan novel paths between locations and follow the path while avoiding obstacles not seen during mapping (Fig.1). We refer the interested reader to the video supplementary material for more examples.

V. Conclusion

In this work, we show that by learning the capability of a local controller, robust and scalable visual topological navigation can be achieved. Due to the simplicity and flexibility of our framework, it can be extended to support non-visual sensors and applied to other robotics problems. Future works include combining multiple sensors to improve the controller, developing better algorithms to resolve ambiguity, improving generalization, and extending to manipulation tasks.

The hyperparameters in our approach are mostly probability thresholds, which are easy to interpret and tune. One important scenario our approach does not handle is when robot deviates too much from all vertices in the navigation graph, where it would fail to find a plausible path. A self-exploratory model can help here, and it can also be used for autonomous map construction.

VI. Acknowledgements

This work was funded in part by ONR grant 63-6094 and by the Honda Curious Minded Sponsored Research Agreement. We thank NVIDIA for generously providing a DGX used for this research via the NVIDIA Robotics Lab and the UW NVIDIA AI Lab (NVAIL).

REFERENCES

[1] N. Savinov, A. Dosovitskiy, and V. Koltun, "Semi-parametric topological memory for navigation," in *International Conference on Learning Representations (ICLR)*, 2018.

[2] Y. Wu, Y. Wu, A. Tamar, S. Russell, G. Gkioxari, and Y. Tian, "Bayesian relational memory for semantic visual navigation," in *International Conference on Computer Vision (ICCV)*, 2019.

[3] A. Kumar, S. Gupta, D. Fouhey, S. Levine, and J. Malik, "Visual memory for robust path following," in *Advances in Neural Information Processing Systems*, 2018, pp. 765–774.

[4] F. Blochliger, M. Fehr, M. Dymczyk, T. Schneider, and R. Siegwart, "Topomap: Topological mapping and navigation based on visual slam maps," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[5] K. Chen, J. P. de Vicente, G. Sepulveda, F. Xia, A. Soto, M. Vazquez, and S. Savarese, "A behavioral approach to visual navigation with graph localization networks," *Robotics Science and Systems (RSS)*, 2019.

[6] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[7] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[8] N. Hirose, F. Xia, R. Martn-Martn, A. Sadeghian, and S. Savarese, "Deep visual mpc-policy learning for navigation," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3184–3191, 2019.

[9] N. D. Ratliff, J. Issac, and D. Kappler, "Riemannian motion policies," *CoRR*, vol. abs/1801.02854, 2018.

[10] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese, "Gibson env: Real-world perception for embodied agents," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 9068–9079.

[11] J. O'keefe and L. Nadel, *The hippocampus as a cognitive map*. Oxford: Clarendon Press, 1978.

[12] T. Hafting, M. Fyhn, S. Molden, M.-B. Moser, and E. I. Moser, "Microstructure of a spatial map in the entorhinal cortex," *Nature*, vol. 436, no. 7052, p. 801, 2005.

[13] C. F. Doeller, C. Barry, and N. Burgess, "Evidence for grid cells in a human memory network," *Nature*, vol. 463, no. 7281, p. 657, 2010.

[14] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, no. 1, pp. 21–71, 1998.

[15] M. J. Milford, G. F. Wyeth, and D. Prasser, "Ratslam: a hippocampal model for simultaneous localization and mapping," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, 2004, pp. 403–408.

[16] B. Kuipers, "The spatial semantic hierarchy," *Artificial intelligence*, vol. 119, no. 1-2, pp. 191–233, 2000.

[17] W. Maddern, M. Milford, and G. Wyeth, "Capping computation time and storage requirements for appearance-based localization with cat-slam," in *2012 IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 822–827.

[18] F. Fraundorfer, C. Engels, and D. Nistér, "Topological mapping, localization and navigation using image collections," in *International Conference on Intelligent Robots and Systems (IROS)*, 2007.

[19] M. Cummins and P. Newman, "Appearance-only slam at large scale with fab-map 2.0," *The International Journal of Robotics Research*, 2011.

[20] C. Linegar, W. Churchill, and P. Newman, "Work smart, not hard: Recalling relevant experiences for vast-scale but time-constrained localisation," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 90–97.

[21] N. Savinov, A. Raichuk, R. Marinier, D. Vincent, M. Pollefeys, T. Lillicrap, and S. Gelly, "Episodic curiosity through reachability," in *International Conference on Learning Representations (ICLR)*, 2019.

[22] S. Bansal, V. Tolani, S. Gupta, J. Malik, and C. Tomlin, "Combining optimal control and learning for visual navigation in novel environments," *Conference on Robot Learning (CoRL)*, 2019.

[23] D. Pathak, P. Mahmoudieh, G. Luo, P. Agrawal, D. Chen, Y. Shentu, E. Shelhamer, J. Malik, A. A. Efros, and T. Darrell, "Zero-shot visual imitation," in *International Conference on Learning Representations (ICLR)*, 2018.

[24] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 7272–7281.

[25] S. Gupta, D. Fouhey, S. Levine, and J. Malik, "Unifying map and landmark based representations for visual navigation," *arXiv preprint arXiv:1712.08125*, 2017.

[26] P. Mirowski, M. Grimes, M. Malinowski, K. M. Hermann, K. Anderson, D. Teplyashin, K. Simonyan, A. Zisserman, R. Hadsell, *et al.*, "Learning to navigate in cities without a map," in *Advances in Neural Information Processing Systems*, 2018, pp. 2419–2430.

[27] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, *et al.*, "Learning to navigate in complex environments," *International Conference on Learning Representations (ICLR)*, 2018.

[28] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "Lqr-trees: Feedback motion planning via sums-of-squares verification," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.

[29] X. Meng, N. Ratliff, Y. Xiang, and D. Fox, "Neural autonomous navigation with riemannian motion policy," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[30] "MIT racecar," 2018. [Online]. Available: https://mit-racecar.github.io/