Robotics: Science and Systems 2019
Freiburg im Breisgau, June 22-26, 2019

A Behavioral Approach to Visual Navigation with Graph Localization Networks

Kevin Chen*, Juan Pablo de Vicente†, Gabriel Sepúlveda†, Fei Xia*,
Alvaro Soto†, Marynel Vázquez‡ and Silvio Savarese*
*Stanford University, Stanford, California 94305, USA
†Pontificia Universidad Católica de Chile, Santiago, Región Metropolitana, Chile
‡Yale University, New Haven, CT 06520, USA

***Abstract*—Inspired by research in psychology, we introduce a behavioral approach for visual navigation using topological maps. Our goal is to enable a robot to navigate from one location to another, relying only on its visual observations and the topological map of the environment. To this end, we propose using graph neural networks for localizing the agent in the map, and decompose the action space into primitive behaviors implemented as convolutional or recurrent neural networks. Using the Gibson simulator and the Stanford 2D-3D-S dataset, we verify that our approach outperforms relevant baselines and is able to navigate in both seen and unseen indoor environments.**

I. Introduction

Despite the ever-changing state of our indoor environments due to rearrangements in furniture, changes in lighting, or the simple accumulation of clutter, humans are able to seamlessly navigate through these dynamic spaces as if nothing in the environment had changed at all. How can we build similar visual navigation systems for robots? Although most approaches for visual navigation today rely on metric maps of the world and precise localization [44], research suggests that biological systems in mice and men rely on coarse spatial layout representations in the form of *cognitive maps* [45]. At the core of such maps, studies suggest that there is a topological description of the environment that can capture relationships about different locations [27, 30, 35, 42]. Animals then execute navigation strategies based on their qualitative knowledge of the space [13]. Motivated by these ideas from psychology research and the success of neural networks at solving a variety of tasks [26], this work revisits early ideas of topological robot navigation [46] and behavioral control [6, 16, 20].

We pose the problem of robot navigation as a graph traversal problem in a topological representation of the environment. More specifically, the goal of the robot is to navigate from place A to place B given a topological map, a plan for how to get from A to B in the map, and the current observation of the environment obtained with an on-board camera (Fig. 1). This formulation leads to three key questions:

1) What is an effective topological representation or map for navigation?
2) How do we localize the agent within this topological representation?
3) Given localization information, how do we control the robot to move according to our plan?

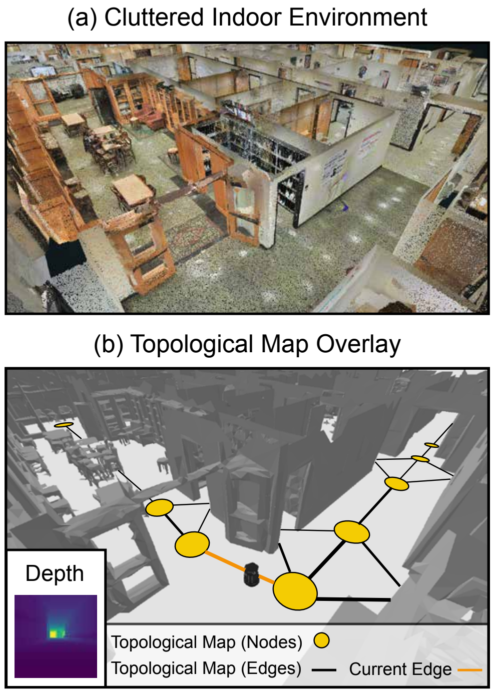

(a) Cluttered Indoor Environment
(b) Topological Map Overlay
Depth
Topological Map (Nodes) ●
Topological Map (Edges) —
Current Edge —

Fig. 1: A robot operates in a cluttered indoor environment (a). Using a topological representation of the environment and depth observations (b), the robot must navigate to its destination, specified as a location within the topological map.

To address question (1), we construct the map to be a directed graph with coarse information about relevant locations for the navigation task (nodes) and connectivity between close locations (edges). Every edge of the map is also labeled with a visuo-motor behavior, such as turn left or turn right, that can be executed to move from the source to the target node.

Given the topological map, we propose using convolutional and graph neural networks to address localization (question (2)). This approach takes advantage of the each neural network architecture. Convolutional neural networks are effective for visual inference [26, 36], and graph neural networks (GNN) capture relational inductive biases [49], making them a natural approach for solving graph-related inference tasks, e.g., as specified by a topological map. We use these models in our approach to infer the location of a robot based on the environment topology and its current observation of the world.

Lastly, the robot must determine how to maneuver itself according to the plan (question (3)). By construction, a path in the topological map can be translated to a navigation plan in the form of a sequence of behaviors. It is therefore trivial to determine which behavior to execute given a localization prediction and a plan. We use neural networks to robustly execute the given behavior, and repeat localization and low-level control at every timestep (e.g. 5 Hz) to ensure smooth transitions to different parts of an environment.

We tested our approach on the Stanford 2D-3D-S dataset [4, 3], which consists of reconstructed meshes of several university buildings with complex layouts and large amounts of clutter. We first constructed and annotated maps of these environments with the proposed topological descriptions. We then incorporated the maps into the Gibson physics-based simulator [48] and extended the simulator's capabilities to create a testbed for benchmarking robot navigation approaches. We contribute to the community our map specification as well as our tools and data, including a dataset that we created for training the learning components of our system.[1] This dataset is composed of 2,371 long sequences of robot observations (e.g., RGB, depth, and semantic data) of the 2D-3D-S buildings and the corresponding robot locations in the topological maps. Using this setup, we show that our method can navigate more successfully than relevant baselines in both seen and unseen cluttered indoor environments.

In summary, the main contributions of our work are:

- We introduce a specification for topological map design in complex, real-world environments.
- We propose a novel framework for localization and navigation using convolutional neural networks in conjunction with graph neural networks. By using a behavioral approach, we are able to robustly navigate through realistic environments.
- We provide a new dataset of robot navigation trajectories in realistic environments with corresponding topological maps.
- We provide a testbed for benchmarking navigation tasks with topological maps using Gibson. Agents can be controlled through the Robot Operating System (ROS), and our evaluation suite allows us to thoroughly analyze performance.

II. Related Work

Our work aims to leverage the advantages of both classical navigation methods and modern deep (DL) learning approaches. First, we advocate for compositionality [1, 2]. As in classical approaches, we separate mapping and localization [44] from path planning and control [28]. Second, we use DL techniques for localization and low-level control. Due to limited space, the next paragraphs focus on reviewing close recent efforts, especially for indoor visual navigation. For extensive reviews on robot navigation, we encourage readers to refer to [25, 44].

Environment representation: Recently, several environment representations have been proposed for navigation. For instance, Parisotto and Salakhutdinov [33] present a Neural Map that emulates a 2D occupancy grid [11]. Mirowski et al. [32] propose loop closure detection to support navigation-relevant representation learning. Moreover, Savinov et al. [38] implement a topological representation where graph edges relate neighboring visual memories. Using this graph, robot localization is performed using a nearest neighbors approach.

Our proposed topological representation takes advantage of the rich semantic structure behind man-made environments and builds directly from Sepulveda et al. [41]. Through this representation, we avoid reliance on metric information [33] or specific robot poses [18]. Our topological maps are significantly sparser than [38]. Different from Sepulveda et al. [41], we consider a reduced set of primitive behaviors for the edges of our topological maps. This facilitates map creation and leads to more generalizable navigation behaviors. Additionally, our approach is more practical than Sepulveda et al. [41] because it does not rely on artificial visual landmarks for navigation.

Localization: A key component of our navigation approach is a localization network that leverages GNNs [5, 40], as a tool to model relational data [49]. To the best of our knowledge, we are the first to use GNNs to pose robot navigation as a graph traversal problem in a topological map of the environment. In robotics, the closest work is Yang et al. [50] that uses graph convolutional networks [24] but in the context of encoding semantic scene priors.

Action space: Many DL approaches are designed for discrete action spaces in order to avoid dealing with low-level mechanics and kinematic constraints. For example, Zhu et al. [52] learn models for target-driven visual navigation and Gupta et al. [17] focus on emulating the Bayesian cycle behind classical SLAM methods [44]. In contrast, the approach that we propose in this work supports continuous action spaces and does not require ground truth odometry information. Our approach reasons at high and low levels of abstraction for planning and motor control, respectively.

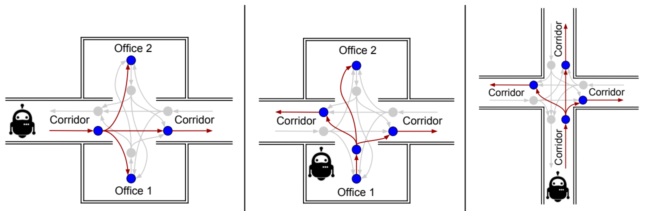
Our work revisits ideas from behavioral robotics for low-level control [6, 20]. This idea can be related to motor control through primitives in manipulation [12, 21, 51], but our goal is not to solve new tasks given demonstrations of an ordered set of actions. Instead, our goal is to execute an abstract navigation plan given a topological map of a realistic human environment.

Evaluation: We test our approach using Gibson, an environment for real-world perception [48] of large-scale indoor spaces [3]. Here, we create topological maps for different spaces and extend Gibson by introducing a benchmark for behavioral robot navigation. With this effort, we support others, e.g., [10, 39, 47], in building a rich and extensible environment for robotics research.

III. Problem Setup

We consider a robot operating in a cluttered indoor environment with the goal of navigating from one node (A) in the topological map to another node (B). In our setup, the agent may not have seen the environment before, so no prior visual information is provided in the map. The ground truth node location A is given to the robot when navigation begins,

[1]Project URL: https://graphnav.stanford.edu.



(a) Entering a room (b) Exiting a room (c) Entering an intersection

Fig. 2: Map sections in the locality of the agent.

but it must rely on its visual input and the map to reach the desired destination. It is crucial for the robot to avoid obstacles – otherwise it will fail the navigation task.

To mimic realistic physical settings, we consider a ROS-controlled Turtlebot robot navigating in the PyBullet[8]-powered Gibson simulator [48] (Fig. 1b). We perform experiments with this robot on environments from the Stanford 2D-3D-S dataset [3, 4]. The dataset was created using a Matterport scanner to capture the geometry of office spaces in three different university buildings. As a result, the floor plans can be very complex and the spaces are filled with clutter including chairs, couches, tables, boxes, and even dollies.

IV. Topological Map Design

Inspired by research in psychology [27, 30, 35, 42], we use a topological representation – a graph – to encode spatial information about the environment. At a high level, each node in the topological map represents a location. Each edge corresponds to a behavior that allows the robot to get from the corresponding source node to the target node, similar to Sepulveda et al. [41]. However, due to the layout complexity of the naturalistic environments in the Stanford 2D-3D-S dataset, we substantially changed the topological map design compared to Sepulveda et al. [41]. In our work, the map annotations are done manually as described in the next paragraphs. Automating this process is a valuable future research direction.

Our insight behind map design is that the robot should be able to traverse to and from any semantic location (e.g. office 1, office 2, pantry 1, conference room 1, etc.) by composing a *minimal length* sequence of behaviors (edges) as specified by the topological map. We leverage the Manhattan world structure of indoor environments [7] and define the behaviors: {*find door (fd), corridor follow (cf), turn left (tl), turn right (tr), straight into room (s)*}. We reduce the specificity of our behaviors (compared to Sepulveda et al. [41]) in order to simplify the design of topological maps, and make navigation more generalizable and transferable across different scenarios. Intuitively, turning left out of an office should require very similar controls to turning left at a four-way intersection/junction or turning left into a room (Fig. 2a).

To avoid localization ambiguities at each position in the topological map, nodes and edges also have associated orientations. For example, if executing a *corridor follow* behavior in a hallway, it is not clear which direction to travel in, since an agent can move along two opposite directions. To resolve this ambiguity, there are two sets of nodes and edges for each corridor, one for each direction (Fig. 2c).

Each room has a corresponding room node that indicates that the robot is within the boundaries of that space and facing any direction. In this case, if the robot is facing towards the inside of the room, a turn behavior (e.g. turn left out of the room) is not well defined. Thus, to ensure smooth transitions in and out of these enclosed spaces, we also add door nodes to rooms. These door nodes indicate that the robot is positioned at the door and oriented towards the exit of the room (Fig. 2b).

Based on these observations, we formulate the topological map as a directed graph. We apply this representation to real world environments using the following rules:

1) Each room in the environment, such as an office or conference room, has its own single node.
2) Each room door also has its own node.
3) The *find door* behavior should connect each room node to its door node.
4) Corridors have two sets of nodes, one for each direction of the corridor.
5) Edges which indicate entering a room should connect to the room node.
6) Exiting a room occurs from the door node.

In general, nodes should be placed at any transition point – that is, any location that may require a change of behavior. For example, upon approaching the exit to a room, there are many possible behaviors to execute, such as *turn left*, *turn right*, or even go *straight* across the hallway into another room. Because this would require a change from the previous behavior (*find door*), a door node should be placed prior to the exit of each room. Likewise, after an agent has turned into a hallway, the robot will likely transition from the turning behavior to a different behavior (e.g. *corridor follow*). Therefore, a node must be placed immediately after the turn to signify the transition. Using this topological map representation, it is then trivial to compute the sequence of behaviors for navigating to and from any location (node) in the map with classical planning algorithms [37].

V. Method

Two challenges for effective navigation with our topological representation are: how to localize the agent, and how to direct the agent along a plan. We organize our navigation approach based on these key problems, as illustrated in Fig. 3.

The first challenge, localization, only needs to be done relative to the topological map. To this end, we propose using a graph neural network in combination with a convolutional neural network (CNN). Once localized, the agent can easily plan paths to any destination in the environment.

The second challenge is how to execute the plan. While the agent continuously updates its localization prediction, it also has to generate motion controls to ensure that it continues to follow the planned trajectory. In our work, for each behavior we implement an individual neural network, which we call a *behavior network*. These networks take as input the current

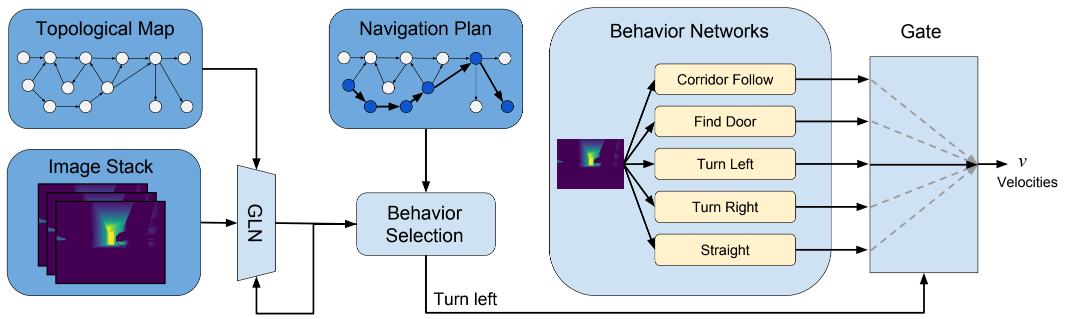


Fig. 3: Our navigation approach addresses localization and behavior selection. Based on the localization estimate from the graph localization network (GLN) and the navigation plan, the agent can select a behavior network for the current timestep. The velocities output from the selected network are used for low-level motor control.

observation of the world – in our case, a depth image – and predict low-level velocity commands.

The above components are combined using a simple behavior selection module, which looks up the correct behavior to execute given the localization prediction and the navigation plan. In the case that a localization estimate is not part of the plan, the module continues selecting the behavior from the last valid position that was part of the plan. Localization and behavior selection are repeated (e.g. at 5 Hz) until the agent reaches its final destination or deviates from the expected path.

Sec. V-A provides a brief introduction to graph neural networks followed by a description of the graph representation used in our model (Sec. V-B). Then, we describe the graph localization network (Sec. V-C), behavior selection (Sec. V-D), and, lastly, the behavior networks (Sec. V-E).

A. Preliminaries on Graph Neural Networks

First introduced by Scarselli et al. [40], GNNs have been shown to be effective at learning relative inductive biases specified by graph structures. The following overview borrows heavily from the description and notation in Battaglia et al. [5].

We define a directed graph to be a tuple $G = (\mathbf{u}, V, E)$, where $\mathbf{u}$ is a global feature for the graph and can be interpreted as a feature representation for the entire graph. $V = \{\mathbf{v}_i\}_{i=1:n}$ is the set of vertices/nodes (cardinality n) where each $\mathbf{v}_i$ is a feature for node i, and $E = \{(\mathbf{e}_k, r_k, s_k)\}_{k=1:m}$ is the set of edge tuples (cardinality m) for which edge k connects the source node with index s_k to the target node with index r_k. For simplicity, we assume the global, vertex, and edge features have the same dimensionality D. That is, $\mathbf{u} \in \mathbb{R}^D$, $\mathbf{v}_i \in \mathbb{R}^D \;\; \forall\, i \in \{1, \ldots, n\}$, $\mathbf{e}_k \in \mathbb{R}^D \;\; \forall\, k \in \{1, \ldots, m\}$.

The basic element of a GNN is a graph network block (GN block). A GN block takes as input a graph $\tilde{G} = (\tilde{\mathbf{u}}, \tilde{V}, \tilde{E})$ and produces an updated graph $\tilde{G}' = (\tilde{\mathbf{u}}', \tilde{V}', \tilde{E}')$ which can have arbitrary feature dimensionality. The GN blocks propagate information encoded in graphs according to their structure.

We compose our GNN with sequential GN blocks such that it eventually computes an output prediction for the task. As detailed in Algorithm 1, the computation is done by first updating the edge features, followed by the node features, and lastly the global features. The update functions $\phi^v(\cdot)$, $\phi^e(\cdot)$, $\phi^u(\cdot)$ and aggregation functions $\rho^{e\to v}(\cdot)$, $\rho^{e\to u}(\cdot)$, $\rho^{v\to u}(\cdot)$ can be implemented in different ways. In our work, we use multi-layer perceptrons for the update functions and summation for the aggregation functions.

B. Graph Representation

To use GNNs for localization, we must convert the concept of a topological map (Sec. IV) into the representation defined in Sec. V-A. Since nodes in the topological maps have semantic room annotations, we can categorize nodes into one of three possibilities: *room*, *hallway*, *open space*. Similarly, each edge is one of five options: *corridor follow*, *turn left*, *turn right*, *find door*, *straight (into room)*. Therefore, we represent each node and edge in the map by a feature vector associated with the node or edge's category, similar to Mikolov et al. [31]. The node and edge features are learned jointly with the graph network and are represented as the embedding lookup table in Fig. 4. The global feature, detailed in Sec. V-C, changes at each timestep and is a function of the current visual input.

C. Graph Localization Network (GLN)

The goal of the graph localization network is to predict the robot's location in the map based on its current visual observation, its last predicted location, and the entire map represented as described in Sec. V-B. To accomplish this, we use

Algorithm 1: Computation in a GN Block

```
function GraphNetwork(u, V, E)
    for $k \in \{1 \ldots m\}$ do
        // update edge features
        $\mathbf{e}'_k \leftarrow \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u})$
    for $i \in \{1 \ldots n\}$ do
        // aggregate incoming edges
        let $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i, k=1:m}$
        $\bar{\mathbf{e}}'_i \leftarrow \rho^{e\to v}(E'_i)$
        $\mathbf{v}'_i \leftarrow \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u})$ // update node features
    // aggregate updated node/edge features
    let $V' = \{\mathbf{v}'_i\}_{i=1:n}$
    $\bar{\mathbf{v}}' \leftarrow \rho^{v\to u}(V')$
    let $E' = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:m}$
    $\bar{\mathbf{e}}' \leftarrow \rho^{e\to u}(E')$
    $\mathbf{u}' \leftarrow \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u})$ // update global feature
    return $(\mathbf{u}', V', E')$
```

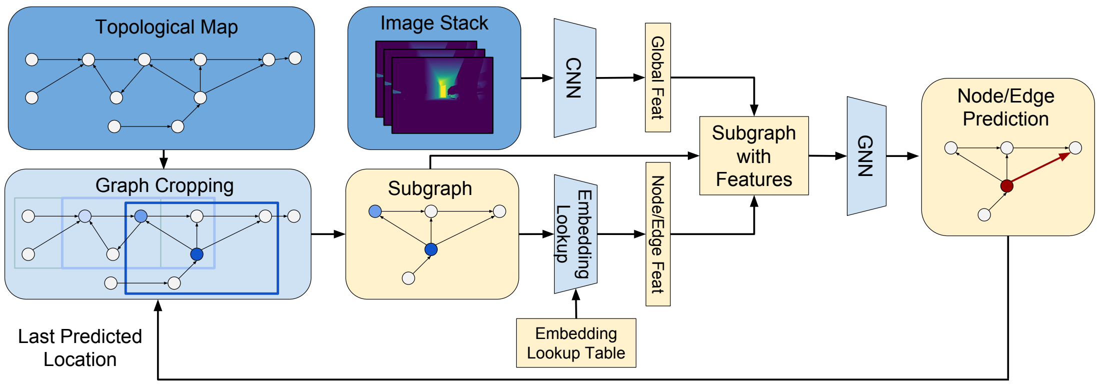


Fig. 4: The graph localization network (GLN) takes three inputs: the depth image stack, the topological map, and the last predicted location. This information is then used to predict the agent's current position within the topological map.

a CNN to process the observations into visual features, which are used as the global feature in our graph representation. In parallel, we crop the graph to the local region around the last predicted location. Then, together with the node, edge, and newly computed global features, the graph is passed through the GNN to predict the agent's current edge in the graph. Note that when navigation begins, the agent is provided with its ground truth location (e.g. office 1). After the initial timestep, the agent relies on its own localization predictions.

More concretely, the inputs to the graph localization network are the graph vertices and edges, the last predicted location, and an image stack I of dimension $H \times W \times C$ where H and W are the image height and width, respectively. To ensure that spatio-temporal information is captured from the visual observations, the agent maintains a stack of the C most recent depth image frames.[2] The graph localization network then processes these inputs as follows:

1) Computing the Global (Visual) Features: The image stack I is forward passed through a convolutional neural network to compute visual features (Fig. 4). These features are used as the global feature $\mathbf{u} \in \mathbb{R}^D$ in our model's GNN.

2) Subgraph Cropping: In parallel to the computation of the global features, the topological map is mapped to its graph representation using the node and edge features described in Sec. V-B. Since the graph of the entire environment can be very large and it is unlikely for the robot to move from one side of the graph to another far side, we crop a local region of the map centered on the previous predicted robot location. In particular, we crop a node if it is above a certain number of edges away from the previous predicted location. The localization prediction is then performed on the local subgraph, which also has the added benefit of reducing computation.

The correct localization is at the center of the subgraph at training time, but this may not necessarily hold true at test time due to noisy (previous) localization predictions. To increase localization robustness, we augment the training data by sampling nearby nodes as the center for the subgraph.

[2]Although other modalities such as RGB may be used instead, we use depth images (clipped to a maximum distance of 3.5 m) to facilitate with generalization to different scenes which may have diverse visual/color appearances.

3) Graph Neural Network Prediction: To train the graph neural network, we treat the localization problem as a classification task. Given a subgraph S with m_s edges, the goal is to classify which of the m_s edges the robot is currently on. In our setup, we use edge classification rather than node classification because edge classification is better defined. For example, at any instant the agent is executing an edge along the navigation plan. In the case of turning left out of an office, it is more natural to claim that the agent is actively executing a turn behavior out of the room than to claim that the robot is still currently located at the room node, even if it has exited the office already. Additionally, the edge carries both source node and target node information, so it is trivial to localize the agent to a (source) node given an edge prediction.

Our network is composed of two sequential GN blocks, with the last block outputting per-edge logits of dimension 1. Let m_s be the number of edges in the subgraph, y be the index of the ground truth edge, and $\mathbf{p}^e$ be the vector of unnormalized probabilities such that each element p_k^e is the unnormalized probability that the agent is on edge k. To train the network, we use a softmax cross-entropy loss on the edge probabilities:

$$l(\mathbf{p}^e, y) = -\log(\exp p_y^e / \sum_k \exp p_k^e)$$

D. Behavior Selection

To determine the agent's current progress in the navigation plan, the edge probabilities output by the GLN are summed at each source node of the topological graph. The highest probability node is then used by the agent to retrieve the behavior that it should execute next. This process avoids the node localization ambiguity, described in Sec. V-C3, while resolving the edge localization ambiguity illustrated by the equally weighted left and right turn behaviors in Fig. 6. For more details, see the supplementary material.

While the model outlined thus far works reasonably well (see Sec. VII), we can improve performance by filtering noisy and multimodal GLN predictions which may be caused by topologically similar intersections. As an example, we explore using a Bayes filter [15] to improve localization.

We implement a particle filter [9, 14] with the assumption that the motion model is unaffected by the control input: $p(x_t|u_t, x_{t-1}) = p(x_t|x_{t-1})$ where x_t is the current location and u_t is the current behavior. For the measurement model $p(z_t|x_t)$, we assume that $p(z_t)$ and $p(x_t)$ are uniform distributions for all timesteps: $p(z_t|x_t) \propto p(x_t|z_t)$. For the transition model, the probability distribution $p(x_t = x_{t-1}|x_{t-1}) = 0.8$ worked well in practice with equally weighted probabilities on the neighbors of x_{t-1} and 0 for all other nodes. We use the output of the GLN to approximate $p(x_t|z_t)$ by summing the outgoing edge probabilities for each node.

E. Behavior Networks

Once the robot has been localized, the next question is how to control the robot given coarse localization information. Unlike most prior deep-learning based approaches, we use a behavioral approach for navigation [41]. Our action space is composed of high-level semantic behaviors which take charge of low-level motor control.

We implement the behavior networks as either a convolutional neural network or a recurrent neural network, depending on the specific behavior. The inputs to each network are visual observations (e.g. depth images) and the outputs are the translational and rotational velocities $\nu = [v_p, v_\theta]$ for the robot. For the *corridor follow* and *find door* behavior, we use a CNN similar to the one used for computing the graph global features in Sec. V-C. For all other behaviors (*turn right*, *turn left*, *straight*), we use a Long Short-Term Memory network [19] with a CNN encoder.

We collected a dataset, as detailed in Sec. VI-A, and trained the behavioral networks via behavioral cloning. We used a mean squared error loss on the predicted and ground truth velocities: $l(\nu, \hat{\nu}) = (\nu - \hat{\nu})^2$.

VI. Experimental Setup

We perform all training and testing in the Gibson simulator, which is powered by the Bullet physics engine [48]. This setup is fairly different from those used by prior DL approaches for navigation. For example, several works use synthetically generated environments which do not accurately represent real indoor spaces [32, 38, 41]. Other approaches have been tested in more realistic house or office settings [17, 18, 47, 50], but they ignore the collision problem entirely and allow the agents to continue their trajectories despite undergoing collisions. In our case, collisions are fatal and result in a failed navigation.

We model the agent as a Turtlebot robot which is operated via ROS. The robot is equipped with a depth camera with a $150°$ field-of-view. This wide angle view alleviates problems with occlusions and doorways. Commands are executed with a frequency of 5 Hz and the robot's velocity is capped at 0.5 m/s. We do not provide ground truth ego-motion to the agent, in contrast to prior works [17, 32, 33].

A. Dataset Collection

We collected data within Gibson in order to train the behavior networks and graph localization network. In particular, we ran thousands of navigation tasks in simulation with the ROS Navigation Stack [29] using ground truth odometry and recorded visual observations from the robot (RGB, depth, and semantic information) as well as odometry information. We also injected noise into the velocity commands during data collection in order to teach the agent to recover from poor positioning. In our experiments, we used only depth images for practicality and generalizability to different environments, but provide RGB and semantic data for future endeavors.

After data collection, we used an automated annotation process based on heuristics to label the robot's trajectory data in relation to the environment's topological map. In particular, we labeled frames/timesteps with tags corresponding to the current behavior that is being executed by the robot, the current node/edge that it is traversing, and the semantic location (e.g., room name). In total, we collected 2,371 motion trajectories with an average of 423.56 frames per trajectory. Data collection is further outlined in the supplementary material.

B. Navigation Evaluation Suite

Because our automated process to label robot trajectories can run in real-time, we were able to create an experimental setup for the systematic evaluation of behavioral navigation approaches. Our setup aims to facilitate reproducibility, such that we can easily perform a thorough analysis of the performance of various navigation models in realistic indoor environments. For example, our setup can identify that a navigation approach is very effective at turning from a hallway into another hallway, but struggles with turning from a hallway into an office. We refer to this infrastructure as our *evaluation suite*, and further detail its key features in the next paragraphs.

1) Generation of Navigation Tasks: Our evaluation suite supports sampling navigation plans (random start and end nodes, followed by the shortest path) for the Stanford 2D-3D-S dataset for which we created topological maps. Sampled plans can then be used to evaluate navigation approaches or roll-outs of navigation policies.

In our experiments, we generated a set of navigation tasks separately from the previously mentioned dataset of Sec. VI-A. Their path lengths range from very short ($<$ 5 meters) to very long ($>$ 50 meters). Example trajectories resulting from these tasks can be observed in Fig. 5.

2) Evaluation Metrics: Our evaluation suite supports several metrics. First, it can evaluate navigation performance based on *success rate*, similar to prior work [39, 41]. A success occurs if the robot can follow the navigation plan all the way until the destination without deviating from the path. Conversely, a failure occurs if the robot deviates or gets stuck due to a collision. Second, our suite can evaluate partial plan completion. More specifically, *plan completion* is measured as the fraction of nodes in the plan that were successfully reached by the agent during navigation. Third, performance evaluation can be conducted in semantically meaningful ways. For example, our suite allows checking the average success rate of a particular *turn* or *junction* in the map, or whether the robot struggles more with turning *into* offices compared to turning *out of* them. These metrics are particularly useful as

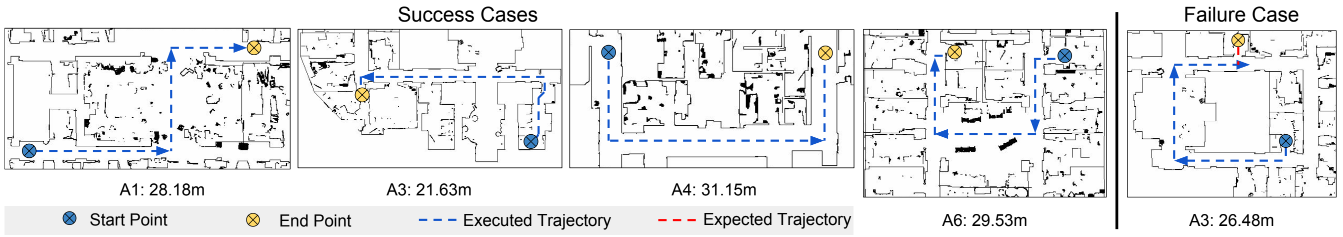


Fig. 5: Examples of executed trajectories by GraphNav with the approximate navigation plan lengths. Seen environments: Areas 1, 6 (A1, A6). Unseen environments: Areas 3, 4 (A3, A4). Best viewed in color.

they provide insight into what kinds of scenarios an approach is likely to succeed or fail in.

C. Implementation Details

We implemented all neural networks using PyTorch [34]. We use depth images of 320×240 pixels and image stacks of the $C = 20$ most recent frames where appropriate. Each behavior network is trained individually. For training all networks, we use the Adam optimizer [23] with a learning rate and batch size of 1e-4 and 32, respectively. We implement the CNNs as a series of strided convolution layers with batch normalization [22], and the graphs are encoded using global, node, and edge features of dimension 512. The supplementary material provides additional implementation details.

VII. Experimental Results

We conduct experiments to evaluate the performance of our navigation approach against several baselines. Following Gupta et al. [18], we use five areas $\{1, 3, 4, 5, 6\}$ from the Stanford 2D-3D-S dataset in our evaluation. Areas 1, 5, and 6 are used for training, area 3 is used for validation, and area 4 corresponds to testing. It is worth noting that there are only three distinct buildings in the dataset: areas 1, 3, 6 correspond to different parts of one building, whereas area 4 and area 5 are captured in two other different locations. Because each area is unique, with varying size and structure, we report results for all of the train, validation, and test areas.

We divide the navigation tasks into three difficulties based on the number of nodes in the corresponding path: 1 through 10 nodes corresponds to difficulty I; 11 through 20 nodes is difficulty II; and > 20 nodes is difficulty III (see the supplementary material for additional details).

Our evaluation considers 3 baselines:

PhaseNet: This network [51] determines when to transition behaviors by predicting their *phase*, or temporal progress. We implement this network with an LSTM trained with a mean-squared error objective on the progress of the behavior.

BehavRNN: Sequence-to-sequence deep learning model [43] trained to perform behavior classification at each timestep with a softmax cross-entropy loss. The model takes as input the current visual observation and the navigation plan (as a sequence of behaviors).

GTL: Navigation approach that uses our automated annotation tools for Gibson to compute the Ground Truth Location (GTL) of the robot in real-time relative to the map. To navigate, the robot executes the behavior network according to its current node in the map. The behavior networks used for this baseline are the same as in our approach.

The first two baselines serve to compare our approach with other relevant DL methods for behavioral navigation. The third baseline helps us study the performance of our behavior networks in isolation from potential localization errors induced by our GLN. We refer to our approach in the following sections as *GraphNav* and *GraphNavPF* (with particle filter).

We evaluate approaches based on full plan success rates, per-behavior success rates, and average plan completion. As mentioned before, plan completion is computed as the fraction of nodes in the plan that the agent successfully reached.

A. Overall Navigation Performance

The quantitative results can be found in Table I. The results show that PhaseNet and BehavRNN models work poorly, resulting in the lowest success rates and plan completion percentages. In contrast, our approach outperforms the PhaseNet and BehavRNN baselines in both seen and unseen areas. For example, in the seen environments (areas 1, 5, 6), we see an average improvement of 819% and 1,471% in the success rate of GraphNavPF over the PhaseNet and BehavRNN baselines, respectively. Since the baselines also utilize the same behavior networks as the GraphNav models, the improvement in performance is primarily due to more accurate localization. While this difference is not as big in the unseen areas, there is still a substantial difference in performance. For example, GraphNavPF achieves a 208% and 325% improvement in success rate, and a 40% and 52% improvement in plan completion compared to PhaseNet and BehavRNN on the validation area. On the test area, the improvement is smaller: 44% (PhaseNet) and 158% (BehavRNN). We suspect this is because the test area is very different from the few available training areas.

Fig. 6 shows a qualitative example of how the localization works with GraphNav. While the agent is in the office, the graph localization network correctly predicts the location. As the robot approaches the door, it is unclear in which direction it will turn. At this point, the network weighs the left and right turns equally, which translates to predicting that the agent is at the door node, triggering a behavior transition. Lastly, as the visuals show the robot turning left, the GLN becomes more confident that it is on the edge corresponding to the left turn.

Successful navigation tasks by GraphNav are shown in the four left-most images of Fig. 5. The robot completes

	Seen Areas (Train)				Unseen Area (Val)			Unseen Area (Test)			
Model	I	II	III	Total	I	II	Total	I	II	III	Total
PhaseNet	12.0 / 52.9	1.6 / 37.3	0 / 21.9	5.3 / 40.7	20.4 / 61.3	0 / 37.8	15.3 / 55.4	17.9 / 55.4	9.3 / 37.2	0 / **24.5**	12.0 / 41.9
BehavRNN	7.6 / 48.1	0 / 26.5	0 / 14.6	3.1 / 32.8	14.8 / 54.9	0 / 40.0	11.1 / 51.1	14.3 / 49.5	2.3 / 27.1	0 / 9.3	6.7 / 34.5
GraphNav	47.0 / 79.9	23.9 / 61.6	18.6 / **61.2**	32.3 / 67.8	40.7 / 75.5	16.7 / 61.2	34.7 / 71.9	25.0 / 67.8	**11.6 / 44.8**	0 / 17.7	16.0 / **52.0**
GraphNavPF	**61.7 / 83.3**	**44.0 / 71.8**	**23.6** / 57.1	**48.7 / 73.8**	**50.0 / 77.6**	**38.9 / 78.1**	**47.2 / 77.7**	**32.1 / 68.7**	9.3 / 41.7	0 / **24.5**	**17.3** / 50.9
GTL†	63.0 / 85.2	53.9 / 79.5	43.2 / 63.7	56.8 / 79.4	74.1 / 86.6	83.3 / 88.9	76.4 / 87.2	57.1 / 77.6	46.5 / 72.0	0 / 33.0	48.0 / 72.0

TABLE I: Performance comparison using success rate (SR) and average plan completion (PC). The † indicates that GTL utilizes additional ground truth information. The best performing entries are bolded per area (not including GTL).

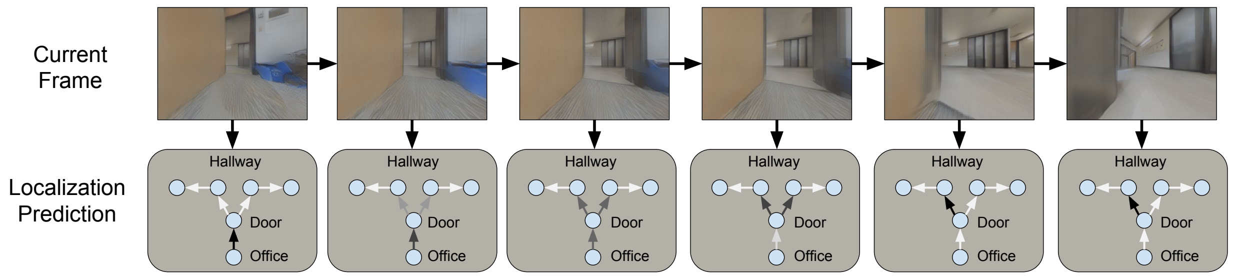


Fig. 6: GLN predictions over time. See the text for more details.

Area ID	cf	fd	tr	tl	s
1 (Seen)	91.3 (438)	98.2 (54)	92.9 (56)	88.5 (61)	25.0 (4)
5 (Seen)	99.8 (523)	93.3 (45)	95.1 (82)	96.6 (88)	100 (4)
6 (Seen)	94.3 (418)	98.2 (54)	83.8 (74)	97.7 (85)	55.6 (9)
3 (Unseen)	96.1 (228)	97.0 (33)	98.2 (54)	92.1 (63)	75.0 (4)
4 (Unseen)	95.7 (376)	87.5 (32)	81.1 (74)	92.5 (67)	- (0)

TABLE II: Average behavior success rate for the GTL model. Number of attempts for each behavior is in parentheses.

trajectories ranging from 20 m to 32 m. On the right is a failure case in which the robot navigated most of the 26 meter-long trajectory but deviated from the path towards the end.

B. Performance of the Behavior Networks

Qualitatively, we observe that the behavior networks used in our approach succeed in their assigned task (e.g., follow a corridor) while being robust to collisions with walls and clutter, especially in structured areas. We refer the reader to the supplementary material for more qualitative examples. We verify our observations quantitatively using the GTL baseline, which uses our behavior networks for motion control along with our annotation tool for localization. As can be seen in Table II, the per-behavior success rates, we observe results generally above 80% and even 90%, indicating robustness in both seen and unseen environments.

C. Limitations

Although our approach significantly outperforms the baselines, there are a few limitations. First, our topological maps were manually annotated and the behaviors were pre-defined. These manual processes limit the scalability of our setup, and the behaviors may not be optimal for navigating in all environments, such as environments with multi-door rooms. In the future, it would be interesting to investigate mechanisms to define topological maps and behaviors in a data-driven fashion.

There is also room for improvement in the navigation success rate (Table I). GTL struggles in certain cases such as large open spaces, which are prevalent in areas 1, 3, 4, and 6. In these spaces, the robot may fail to orient itself correctly and walk into a corner or deviate from the correct path. Recovery is difficult because the clutter prevents the agent from having a direct line-of-sight to the room exit and because the depth camera has a maximum depth 3.5 m.

A second source of error comes from imperfect localization and therefore imprecise behavior transition timing. When this happens, the agent either (1) selects the wrong behavior, or (2) transitions behaviors slightly too early or too late, causing the robot to deviate from the intended path especially since the agent does not anticipate its next behavior.

VIII. Conclusion

We introduced an effective topological map design for behavioral navigation and, to the best of our knowledge, are the first to propose graph neural networks for robot localization. We tested our proposed approach using Gibson and provide an open-source testbed for benchmarking navigation in complex, human environments. Our results show the potential of combining DL with classical robotics, and we hope that our work inspires further research in visual navigation.

IX. Acknowledgements

The Toyota Research Institute ("TRI") and the National Science Foundation (NSF) Graduate Research Fellowship (Grant No. DGE 1147470) provided funds to assist with this research. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of TRI, any other Toyota entity, or the NSF. This work is also partially funded by Fondecyt grant 1181739, Conicyt, Chile.

References

[1] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48, 2016.

[2] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 166–175. JMLR. org, 2017.

[3] I. Armeni, A. Sax, A. R. Zamir, and S. Savarese. Joint 2D-3D-Semantic Data for Indoor Scene Understanding. *ArXiv e-prints*, February 2017.

[4] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2016.

[5] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[6] Rodney Brooks. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, 2(1):14–23, 1986.

[7] James M Coughlan and Alan L Yuille. Manhattan world: Compass direction from a single image by bayesian inference. In *Proc. of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 941–947. IEEE, 1999.

[8] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2018.

[9] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots. In *ICRA*, volume 2, pages 1322–1328, 1999.

[10] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[11] A. Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. PhD thesis, Carnegie Mellon University, 1989.

[12] J. Felip, J. Laaksonen, A. Morales, and V. Kyrki. Manipulation Primitives: A Paradigm for Abstraction and Execution of Grasping and Manipulation Tasks. *Robot. Auton. Syst.*, 61(3), March 2013.

[13] Patrick Foo, William H Warren, Andrew Duchon, and Michael J Tarr. Do humans integrate routes into a cognitive map? map-versus landmark-based navigation of novel shortcuts. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 31(2):195, 2005.

[14] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *AAAI/IAAI*, 1999(343-349): 2–2, 1999.

[15] V. Fox, J. Hightower, Lin Liao, D. Schulz, and G. Borriello. Bayesian filtering for location estimation. *IEEE Pervasive Computing*, 2(3):24–33, July 2003. ISSN 1536-1268. doi: 10.1109/MPRV.2003.1228524.

[16] Matthias O Franz and Hanspeter A Mallot. Biomimetic robot navigation. *Robotics and autonomous Systems*, 30 (1-2):133–153, 2000.

[17] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive Mapping and Planning for Visual Navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[18] Saurabh Gupta, David Fouhey, Sergey Levine, and Jitendra Malik. Unifying map and landmark based representations for visual navigation. *arXiv preprint arXiv:1712.08125*, 2017.

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[20] Ian Horswill. Polly: A vision-based artificial agent. In *AAAI*, pages 824–829, 1993.

[21] De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese, and Juan Carlos Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. *arXiv preprint arXiv:1807.03480*, 2018.

[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[24] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[25] I. Kostavelis and A. Gasteratos. Semantic mapping for mobile robotics tasks: A survey. *Robotics and Autonomous Systems*, 66(103):86–103, 2015.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999134.2999257.

[27] Benjamin Kuipers and Yung-Tai Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and autonomous systems*, 8(1-2):47–63, 1991.

[28] J.C. Latombe. *Robot motion planning*. Kluwer Academic, 1991.

[29] David V Lu, Dave Hershberger, and William D Smart. Layered costmaps for context-sensitive navigation. In

Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on, pages 709–715. IEEE, 2014.

[30] Kevin Lynch. *The image of the city*, volume 11. MIT press, 1960.

[31] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[32] 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. *arXiv preprint arXiv:1611.03673*, 2016.

[33] Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*, 2017.

[34] 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.

[35] J. Piaget and B. Inhelder. *The Child's Conception of Space*. Norton, 1956.

[36] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[37] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

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

[39] Manolis Savva, Angel X. Chang, Alexey Dosovitskiy, Thomas Funkhouser, and Vladlen Koltun. MINOS: Multimodal indoor simulator for navigation in complex environments. *arXiv:1712.03931*, 2017.

[40] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[41] Gabriel Sepulveda, Juan Carlos Niebles, and Alvaro Soto. A deep learning based behavioral approach to indoor autonomous navigation. *arXiv preprint arXiv:1803.04119*, 2018.

[42] Alexander W Siegel and Sheldon H White. The development of spatial representations of large-scale environments. In *Advances in child development and behavior*, volume 10, pages 9–55. Elsevier, 1975.

[43] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

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

[45] Edward C Tolman. Cognitive maps in rats and men. *Psychological review*, 55(4):189, 1948.

[46] Olivier Trullier, Sidney I Wiener, Alain Berthoz, and Jean-Arcady Meyer. Biologically based artificial navigation systems: Review and prospects. *Progress in neurobiology*, 51(5):483–544, 1997.

[47] Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. Building generalizable agents with a realistic and rich 3D environment. *arXiv preprint arXiv:1801.02209*, 2018.

[48] 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.

[49] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[50] Wei Yang, Xiaolong Wang, Ali Farhadi, Abhinav Gupta, and Roozbeh Mottaghi. Visual semantic navigation using scene priors. *arXiv preprint arXiv:1810.06543*, 2018.

[51] Tianhe Yu, Pieter Abbeel, Sergey Levine, and Chelsea Finn. One-shot hierarchical imitation learning of compound visuomotor tasks. *arXiv preprint arXiv:1810.11043*, 2018.

[52] 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.