GraphMapper: Efficient Visual Navigation by Scene Graph Generation

Zachary Seymour, Niluthpol Chowdhury Mithun, Han-Pang Chiu, Supun Samarasekera, Rakesh Kumar
Center for Vision Technologies, SRI International, Princeton, NJ; Email: firstname.lastname@sri.com

***Abstract*—Understanding the geometric relationships between objects in a scene is a core capability in enabling both humans and autonomous agents to navigate in new environments. A sparse, unified representation of the scene topology will allow agents to act efficiently to move through their environment, communicate the environment state with others, and utilize the representation for diverse downstream tasks. To this end, we propose a method to train an autonomous agent to learn to accumulate a 3D scene graph representation of its environment by simultaneously learning to navigate through said environment. We demonstrate that our approach, GraphMapper, enables the learning of effective navigation policies through fewer interactions with the environment than vision-based systems alone. Further, we show that GraphMapper can act as a modular scene encoder to operate alongside existing Learning-based solutions to not only increase navigational efficiency but also generate intermediate scene representations that are useful for other future tasks.**

I. Introduction

Automatic understanding of geometric relationships among semantic objects in the perceived scene is a key ability for humans to navigate in new environments. Humans reason about the 3D topology of recognized objects in the space to enable efficient navigation, such as finding the shortest path from one room to another avoiding obstacles. Human brains automatically builds a unified representation of the environment to support memory and guide future actions for navigation tasks.

The goal for this paper is to train an autonomous agent to resemble this human capability for efficient visual navigation. Training agents with intermediate scene representations ought to enable the agents to learn faster and are more robust under different situations [1]. Recently, deep reinforcement learning (DRL) methods demonstrated promising results for autonomous agents on navigation tasks, by learning a direct mapping from observations to actions through trial-and-error interactions with its environments. However, most of these methods are purely data driven without constructing intermediate representations. Generating a powerful but cost-effective scene representation, that encodes both semantic objects and their geometric relationships, is still a challenging and unsolved problem.

We propose to learn and to generate "scene graphs" for fulfilling this purpose. Scene graphs are compact representations of the relationships between the objects in a scene or image, using a graph data structure. Each node in the graph represents an instance of an object in the scene, and each edge encodes the relationship between a pair of objects. Scene graphs offer an appealing mid- to high-level representation of the scene that are easily digestible by many applications, and graph-structured data in general has been shown to act as a strong prior to regularize and refine features in neural networks [2]. We adapt the recently introduced Graph Transformer Network (GTN) [3] to the task of simultaneous scene graph generation and visual information refinement for building up powerful scene representations to perform visual navigation, which we refer to as GraphMapper (Figure 1). Beginning with a candidate set of scene objects and simple, pairwise edges defining the geometric relationship of each, we learn sparse, structural scene graph representations that not only improve the sample efficiency of the learning policy but also afford many opportunities for down-stream tasks and future applications. The contributions of proposed GraphMapper, are as follows:

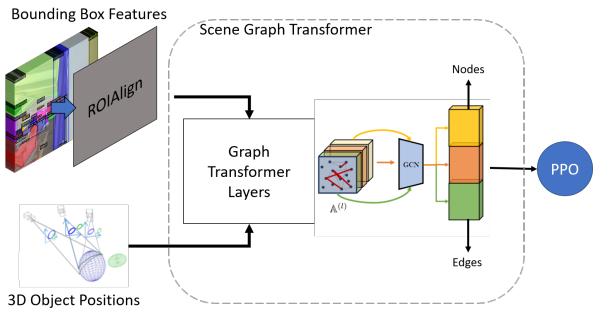

Fig. 1: An overview of proposed GraphMapper. Given bounding boxes of scene objects and estimates of their 3D positions, our Scene Graph Transformer Network emits a graph representation of the scene, along with node- and edge-level features, to enable visual navigation.

1) We are the first such work to explore the generation of 3D scene graphs as an additional output alongside training an autonomous agent to perform visual navigation.
2) We offer a widely-applicable and easily-adaptable "renderer-supervised" approach that alleviates the need for large-scale, annotation of scene graph ground truth.
3) We show how the graph-structured data refines the agent's visual observations, leading to greater performance over the same training regime on practical learning tasks, i.e., PointGoal and Vision Language navigation (VLN).
4) Finally, we explore several representative examples of the scene representations the agent learns to predict that hint at several interesting uses in future applications.

II. Related Work

We briefly review prior works related to different learning-based approaches to the problem of autonomous navigation, with the discussion on scene representations in these methods.

Learning-based Approaches: The traditional (non-learning) approach decomposes this problem into two steps, i.e., mapping [4], [5], [6], [7] and path planning [8], [9], [10]. Instead of dividing the problem into two discrete stages, learning based methods [11], [12], [13], [14], [15], [16], [17], [18] address it by learning direct correspondences from sensor data to robot motion commands as navigation policies. Due to the difficulty of training agents through interactions with real world, most of the methods focus on training using photo-realistic simulators [19], [20], [21], [22]. However, this requires significant training data (experience) to implicitly learn the knowledge to achieve better navigation results [20], [19], [23] compared to traditional methods in new environments. Moreover, it lacks capabilities to explain or reason its behaviors or actions. While recent works have shown promise in leveraging a hierarchical policy combined with an analytical planner [18], [24], [25], our focus remains on improving the efficiency of end-to-end learning based solutions by improving intermediate representations.

Scene Representations: There are recent works [15], [26], [27], [18], [28] that aim to combine the strengths from traditional SLAM-based navigation methods and data-driven learning-based methods. These works build up some forms of top-down occupancy map of the environment, which the agent either uses as an additional feature for its policy, or as an environmental proxy to select intermediate goals, or both. By utilizing this occupancy map, the agent is aware of its options [29], [30] to explore more areas for navigation tasks. Building this kind of representation helps the agent to learn more effectively, by increasing the sample efficiency of learning the policy and reducing the amount of training data for visual navigation [18], [15], [21], [31]. While utilizing such a representation in deep reinforcement learning (DRL) or imitation learning (IL) based methods has similar benefits that a geometric map typically supports in traditional SLAM, it still lacks the cognitive capability that the humans possess to understand the 3D topology among semantic objects in the environment. As such, we seek to develop a richer, sparser scene representation specifically for autonomous agents in navigation tasks. Our representation is in the form of a scene graph that encodes entities, semantic objects, and their relationships in the environment. Of note, we distinguish our work from those in visual navigation which utilize or construct environment-level topological graphs [32], [33], [34], [35] in that our focus is on generating a scene-level graph from a particular viewpoint.

Scene Graph Generation and Graph Neural Networks: Scene graph representations serve as a powerful way of representing image content in a heterogeneous graph, in which the nodes represent objects or regions in an image and the edges represent relationships between the objects. Scene graphs have largely been studied as a higher-level, more descriptive form of image annotation (see [36]); however, they have also been recently studied in other applications, such as image synthesis [37], [38], [39], [40]. Similarly, graph neural networks (GNN)—of which graph convolutional networks (GCN) [41] seem to be the most popular type—are a new class of neural networks that allow the processing of graph-structured data in a deep learning framework. Such networks work by learning a representation for the nodes and edges in the graph and iteratively refining the representation via "message passing;" *i.e.*, sharing representations between neighboring nodes in the graph, conditioned on the graph structure.

The majority of works on scene graph generation focus on 2D scenes: given a set of objects in an image, the goal is to either predict a relationship between object pairs (*cf.*, [42], [43]) or to transform a complete graph of the scene into a sparser set of semantic relationships (*cf.*, [44], [45], [46]). A few recent works instead pose the scene graph generation problem in a less anthropocentric way: they focus on finding graphs describing the geometric structure of the scene, often in 3D [47], [48], [49]. However, such methods generally require learning a separate function for each type of edge in order to perform message passing in a manner that treats different types of edge relationships differently. The computation becomes expensive when many nodes or edge types exist, and the GNN computation can become too heavy to act as a component in a larger system. To enable end-to-end graph generation and graph feature learning, we leverage Graph Transformer Networks [3] to build up complex edges, by forming "meta-paths" from given an initial set simple geometric edge relationships. This network learns to classify nodes and edges, while performing feature message passing using lightweight GCN operations.

III. Problem Setup

To verify and demonstrate GraphMapper, we train our proposed architecture for PointGoal and VLN tasks. We assume our agent is equipped only with `RGB` camera and a `Depth` sensor, in addition to some means of obtaining region of interest (ROI) proposals for its current view. Here we follow [49] that utilizes semantic masks provided by the simulator's rendering engine as ROIs. In practice, these ROIs can be predicted on the fly by, *e.g.*, an object detector [50].

Our goal is to learn an action policy π for the agent to utilize `RGB` and `Depth` sensor data in a more effective manner, that enhances scene understanding and enables efficient navigation. To achieve this, we propose a new method that enables the agent to generate single-shot 3D scene graphs of the environment using a novel formulation of the recently-introduced Graph Transformer Networks [3]. In other words, by navigating through or exploring the environment, the agent learns to generate a graph representation from its current observation. The graph representation encodes the semantic objects in the scene and the 3D structural relationships among them. Prior work has shown that providing geometric information of the environment (such as a top-down occupancy map) helps the autonomous agent to learn more efficiently, to transfer more easily across domains, and to explore new environments more aggressively [15], [18]. Our scene graph representation models objects in the scene along with their geometric relationships. It fuses both the semantic entities and their geometry information into a single, computationally-efficient representation. This representation enables higher-level scene understanding that cannot be easily achieved with a top-down geometric map.

IV. Renderer-Supervised Scene Graph Generation

There has been several recent works addressing scene graph generation; *i.e.*, given one or more views of a scene, output a graph $G = (V, E)$ where V, the set of nodes, represents objects present in the scene and E, the set of edges, the relationships between them. One issue in adapting these methods for visual navigation tasks, is that a number of these methods 1) rely on a large quantity of manually-annotated, graph level supervision to train [45], [44], [49], 2) output a large set of semantic relationship edges, many of which are meaningless in the context of autonomous agents (*e.g.*, "belonging to," "wearing," "playing") [46], [45], or 3) have only been shown to work either in synthetic environments or in narrow fields-of-view unsuitable for visual navigation [49], [47].

By contrast, we seek a solution for the generation of scene graphs that does not require any additional annotation, outputs only structurally-meaningful relationships, and operates on the fly from the point-of-view of an autonomous agent. Our core intuition is that in the typical training environment for such an autonomous agent—*e.g.*, Habitat [19], Gibson [51], MINOS [52], or AI2-THOR [16]—there is a rich quantity of underlying data that goes into rendering the agent's current view. Namely, given a view frustum derived from the agent's current pose (3D position and 3D orientation or angular heading) and camera extrinsics, the agent's `RGB` observation is a slice of the underlying 3D mesh, which readily encodes structural information about the entire scene, often augmented with discrete object meshes. We propose to use this information for "renderer-supervised" scene graph generation.

Dataset Extraction. First, we introduce our method for automatically extracting scene graph ground truth to supervise our scene graph generation model. To do so, we make use of the underlying semantic annotations that are readily available in most of the popular learning environments [51], [19]. Such annotations, propagated from semantic point cloud or existing 3D models, are cheaper to collect than full annotations for large-scale environments [48]. The objects defined in this scene become the nodes in our graph. To provide structurally-meaning edges to the graph, we utilize the information already encoded in the 3D mesh. Inspired by prior work on 3D scene graphs [49], [47], [48] and the types of edges used in other domains (*e.g.*, Visual Genome [53]), we utilize two main types of edges: *co-planarity*, with a three types of edge denoting two objects are approximately co-planar in the x, y, and z directions; and *same region*, where an edge represents that two objects share a region in the semantic scene (typically indicating "same room," depending on the discretization of the space). As we utilize the underlying 3D mesh and the environment's semantic scene, we can directly compute either oriented or axis-aligned bounding boxes (AABB) for objects in the environment, which makes computing this ground truth simple. We find empirically that it adds relatively little time to the rendering of the environment, compared to loading semantic 3D mesh from disk.

Graph-based Scene Observations. Now, we discuss the input to our scene graph generation network that maps observations to the ground truth scene graphs. As mentioned above, our agent's visual observation begins with a set of ROIs given its `RGB` observation. Each ROI corresponds to one of the objects defined in the given scene; namely, one of the defined Matterport object classes. While earlier methods approach the problem of scene graph generation by predicting the existence of and label for an edge between each pair of objects independently, it has become common practice to construct a complete graph between all of the object nodes and utilize message passing to label the edges holistically. That is, for a set of N ROIs, a graph will be generated with $N(N-1)$ candidate edges. As our graph relationships are essentially structural in nature rather than linguistic or semantic, we use the relative pose and dimensions of the observed objects as initial features for the candidate edges, similar to the quadric representation used in [49]. We utilize the agent's depth observation to estimate both the dimensions of each object, as well as the approximate location of the object's front plane in the agent's coordinate frame. We concatenate these two 6-dimensional vectors (3 dimensions and 3 coordinates) for each pair of nodes as the input edge representation. As there is no assumed directionality to our edges (and thus no strict notion of "subject" or "object" in the predicted relationships as in prior work [45]), we adapt a newly-introduced GNN formulation to simultaneously extract meaningful edges from this fully-connected graph and to perform message passing across heterogeneous edge types.

A. Scene Graph Transformer Network

Most prior scene graph generation algorithms are built around the idea of iterative message passing [44], whereby a set of node features and candidate edge features are repeatedly propagated over the edges of the graph to eventually output the final relationship predictions. However, because our goal is to learn to predict the graph edges while simultaneously utilizing the graph structure for performing the agent's downstream task, we cannot realistically afford such a complex operation. Furthermore, because we leverage graph input extracted from the rendering engine rather than higher level annotations, the relationships represented by single-hop graph edges may be too simple for the node representations to contain meaningful global information. Based on these two insights, we leverage Graph Transformer Networks (GTN) [3] as a model that can perform soft edge selection to represent composite relationships and handle the high-degree of heterogeneity present in our graph. GTNs [3] were introduced as a graph variant of the Spatial Transformer Network [54] to automatically learn transformations of the graph adjacency matrix to encode "meta-paths" (paths comprised of heterogeneous edge types) between nodes. This transforms the input graph structure into one defined by these meta-paths of arbitrary length and edge type in a way that is more suitable for the downstream task. We provide a brief summary of the model below.

Notation. The input to our Scene Graph GTN is a complete graph $G = (V, E)$ on the set of V nodes. If $N = |V|$ denote the number of nodes, then $E = \{(i, j) \mid \forall i, j \in V\}$ and $|E| = N(N-1)$. Each $v \in V$ is represented by a vector

$z \in \mathbb{R}^{d_V}$, where d_V is the output dimension of CNN used to process the node ROIs. Each $e \in E$ is represented by a vector $a \in \mathbb{R}^{d_E}$, where d_E is the number of input edge features defined above; *i.e.*, $d_E = 6$. The structure of the input graph can then be compactly represented as tensor $A \in \mathbb{R}^{N \times N \times d_E}$. After several layers of processing, the output of this network is a set of N labels, one for each node in the original graph, and a new, sparse adjacency matrix $A_s \in \{0, 1\}^{N \times N}$. In experiments, we use three (3) layers for both the adjacency matrix in GTN and for node-node message passing in the following GCN.

Graph Transformer Layer. The generation of new set of paths is performed by a sequence of Graph Transformer (GT) layers, each of which uses soft attention to select candidate graph structures initial complete adjacency matrix A. The layer outputs a new multi-hop adjacency matrix as the composition of the soft-selected candidate graph structures. It computes a transformation of adjacency matrix A using a 1×1 convolution with the softmax of two sets of learned weights $W_1, W_2 \in \mathbb{R}^{1 \times 1 \times d_E}$ (where W_1 and W_2 parameterize the convolution operation) in a manner similar to Transformer's dot-product attention [55]. A number of such layers can be stacked to generate arbitrary length l paths with adjacency matrix A_P:

$$A_P = \left(\sum_{t_1 \in \mathcal{T}^e} \alpha_{t_1}^{(1)} A_{t_1} \right) \left(\sum_{t_2 \in \mathcal{T}^e} \alpha_{t_2}^{(2)} A_{t_2} \right) \cdots \left(\sum_{t_l \in \mathcal{T}^e} \alpha_{t_l}^{(l)} A_{t_l} \right) \tag{1}$$

To ensure the network is able to learn meta-paths that are of any length and that may also include some of the original graph edges, the identity matrix is always included; *i.e.*, $A_0 = I$.

Graph Convolutional Network. Following a sequence of Graph Transformer layers, a graph convolutional network (GCN) [41] is used to learn useful representations for node representations in an end-to-end fashion. With $H^{(l)}$ the node features at the lth layer in the GCN, the message passing operation is performed as:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)} \right), \tag{2}$$

Here $\tilde{A} = A + I \in \mathbb{R}^{N \times N}$ represents A with added self-loops; $\tilde{D}$ is the degree matrix of $\tilde{A}$ (where D_{ii} represents the degree of node i and $W^{(l)} \in \mathbb{R}^{d \times d}$ is a learned parameter matrix for layer l). In standard GCN, only the node-wise linear transform $H^{(l)} W^{(l)}$ is learnable. However, the preceding GTN layers output several update (meta-path) adjacency matrices at each iteration, allowing the entire graph structure to be learned. If the number of output channels of 1×1 convolution in each GT layer is C, then each layer outputs a pair of intermediate adjacency tensors Q_1 and $Q_2 \in \mathbb{R}^{N \times N \times C}$. This architecture then functions as as an ensemble of GCNs operating on each of C adjacency matrices output by the final GT layer. A GCN is applied to each channel C and the multiple node representations are concatenated to form a holistic node representation $Z \in \mathbb{R}^{N \times C \cdot d}$, where d is the node feature dimension.

For our purposes, we also preserve the output adjacency tensor $Q_1 \otimes Q_2 = \hat{A} \in \mathbb{R}^{N \times N \times C}$ as an C-dimensional feature vector for each edge. We then utilize two separate fully-connected heads. It first predicts a class for each node given Z, using standard cross-entropy given the objects present in the scene as supervision. The other outputs zero or more types for each edge given $\hat{A}$, using multi-label binary cross-entropy given the edge types extracted from the renderer (as described in §IV) as supervision. The whole architecture also receives feedback from gradients propagated from the reward signal of the DRL or IL policy learning algorithm.

V. Evaluation

In this section, we demonstrate the effectiveness of our proposed GraphMapper, for producing refined, structural visual representations for DRL-based PointGoal (V-A) and IL-based VLN navigation (V-B) tasks. During navigation, the agent constructs and accumulates a scene graph representation for the perceived environment for more effective decision-making. Our scene graph representation also supports novel downstream applications, such as explaining or reasoning its behaviors or actions. Although we choose PointGoal and VLN as target tasks, our GraphMapper can be applied for any navigation task [56] that requires the agent to move throughout the environment, such as ObjectGoal, AreaGoal, or Exploration. Following prior works [56], [57], [58], we use Success weighted by Path Length (`SPL`) and Success Rate (`SR`) as the primary metrics to evaluate the navigation performance of agents. We also report Normalized Dynamic Time Warping (NDTW) and Navigation Error (NE) for VLN agents [59], [58]. Please see supplementary for more details on experiments, and qualitative applications.

A. PointGoal Navigation

PointGoal navigation requires the agent to navigate to an assigned target position, given a starting location and orientation. Our base network for performing PointGoal navigation is derived from the "RL + Res18" baseline described in [18]. Namely, we replace the ResNet18 [60] visual encoder with our Scene Graph Transformer Network. The features of each node are average pooled and passed through fully-connected layer, and then input along with the agent's previous action and an encoding of the target to a one-layer GRU [61] to enable the agent to maintain coherent action sequences across several time steps (the actor), followed by a linear layer (the critic) to predict the next action. To increase sample efficiency, we follow [18] and replace the continuous representation of the agent's relative angle and distance to the target and the one-hot encoding of its previous action with a learned 32-dimensional embedding of the same. We bin relative distance into bins increasing in size exponentially with distance and relative angle into 5 degree bins before passing through embedding layer.

The network is trained end-to-end using proximal policy optimization (PPO) [62], following [19]. All policies trained with this receive a reward R_t at time step t proportional to the distance they have moved towards the goal. We also include an auxiliary coverage reward equal to the change in the percentage of the ground truth scene graph observed by agent. This helps to encourage the agent to explore and seek better views to complete more of scene graph. For more training details (*e.g.*, learning rates, model architectures, *etc.*), we refer to [18], [19].

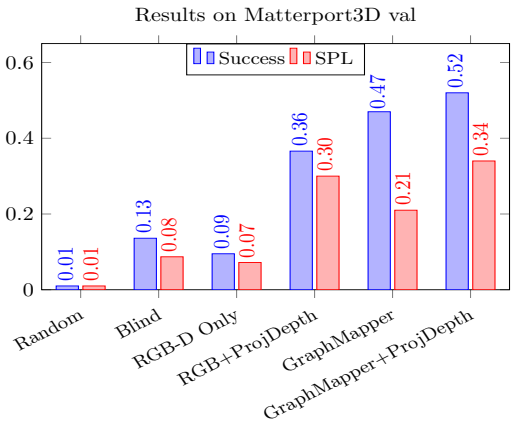

Fig. 2: A comparison of several RL-based PointGoal solutions, with and without our proposed GraphMapper scene graph encoder.

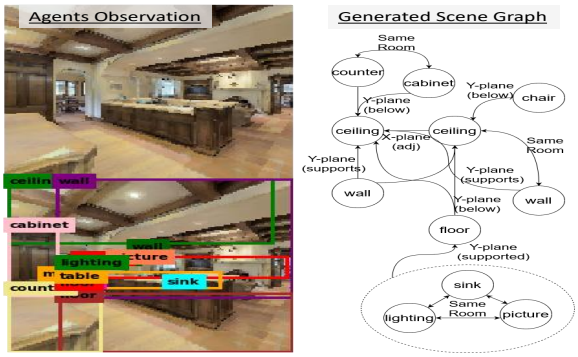

Fig. 3: Example of a scene graph generated by our method. We show the agent's observation (left top), the predicted node labels with their original bounding boxes (left bottom), and the generated graph (right).

Evaluated Models. To evaluate our network, we modify and compare against two best-performing, end-to-end RL methods [18], i.e., **(1) RL + GraphMapper:** An RL policy that replaces the ResNet18 [60] of "RL + Res18" [18] with our GraphMapper policy architecture, followed by a GRU [61]. **(2) RL + GraphMapper + ProjDepth:** A baseline adapted from "RL + Res18 + ProjDepth" [18], [15] that—in addition to replacing ResNet18 visual encoder with GraphMapper—also projects the depth to an egocentric top-down map as input to the RL policy. Following [18], we pass the depth image through a three-layer CNN [19] before concatenating with output of GraphMapper and passing to the GRU policy.

For fair comparison (as explained below), we also implement an RGB-D only baseline that uses ResNet50 identical to backbone of GraphMapper (and used in works such as [23]). We believe our results demonstrate GraphMapper can act as a modular component in any learning-based visual navigation system. We hope to demonstrate the compatibility of ours with other approaches (e.g., Active Neural SLAM [18]) in future.

Quantitative Results. In Figure 2, we present the results of several Reinforcement Learning (RL) models. In addition to the Random (unlearned) and Blind (no visual observation) baselines, we show two visual navigation policies without (RGB-D Only and RGB+ProjDepth) and with (GraphMapper and GraphMapper +ProjDepth) our Scene Graph Transformer Network in the visual encoder. These results were calculated on the Matterport3D validation split [19], a set of 495 navigation episodes not observed during training. All agents were trained with an equal amount of experience—14 million steps on the Matterport3D train split—which has been shown to be sufficient experience for (depth-only) learning-based models to outperform traditional SLAM [19]. From Figure 2, with this amount of training the ResNet50 model underperforms compared to the Blind agent. While other works have shown that such architectures are able to perform quite well after significantly more training [23] (99% SR after training for 2.5 billion observations), we observe even after 14 million observations of the environment, the policy has not quite learned to effectively generate feature representations useful for navigation. We show that addition of GraphMapper leads to more efficient training, by comparing agents trained with an equal amount of experience, also leading to significant performance improvements on the given training budget.

On the other hand, RGB+ProjDepth, which supplements its visual observation with an accumulated, top-down egocentric map of the environment, is able to overcome these deficiencies and perform quite well. Replacing the standalone ResNet50 encoder with GraphMapper yields even greater improvements in terms of success, although its path efficiency, as measured by SPL, decreases. It may be that the graph-based exploration reward alone overbiases the model towards taking longer paths, focusing on observing more of the graph structure. We note, however, that although the backbone network of GraphMapper encodes similar visual priors (from pretraining on ImageNet and then MSCOCO object detection) as the RGB-D Only method, our Scene Graph Transformer Network is better able to refine the visual features with semantic-structural priors to increase the sample efficiency of the network, nearly doubling the SPL with the same amount of experience. Furthermore, GraphMapper +ProjDepth yields the best results, as structurally-refined visual features and egocentric map features are more closely aligned in encoding the agent's local 3D space, such that the policy can take effective advantage of each.

Qualitative Results. In addition to effectively learning visual navigation policies, GraphMapper also enables the generation of scene graphs of the environment. In Figure 3, we present an example of a scene observed by the agent from Matterport3D and the scene graph generated internally by GraphMapper. For each node classified with high confidence ($p > 0.5$), we draw bounding box provided to the network and the label predicted. On the right of Figure 3, we draw the same nodes as part of the heterogeneous scene graph with a subset of the most confident evidence, chosen by high attention scores (top 10%) from the meta-path selection in the Scene Graph Transformer Network. We label each edge with its predicted category (either "same room" or sharing one of the x, y, or z planes) as well as, where appropriate, a potential semantic explanation of said relationship. For example, as in prior works [49], [47], objects that are relatively co-planar both horizontally and vertically can be said to share a "supporting–supported by" relationship. Furthermore, by accumulating these

TABLE I: Several VLN agents on VLN-CE validation-seen set.

Learning	Model	Evaluation Metrics			
		SR↑	**SPL↑**	**NDTW↑**	**NE↓**
No	Random [58]	0.02	0.02	0.28	10.20
	Hand-Crafted [58]	0.04	0.04	0.33	9.56
Yes	RGB [58]	0.03	0.03	0.29	10.76
	Depth [58]	0.24	0.23	0.46	8.55
	RGB-Depth [58]	0.25	0.24	0.45	8.54
	GraphMapper	0.18	0.17	0.40	8.82
	GraphMapper-Depth	**0.27**	**0.26**	**0.47**	**8.37**

graphs over an entire navigation trajectory, GraphMapper can be utilized for richer downstream tasks, such as mapping the environment into discrete semantic spaces (see supplementary).

B. Vision-Language Navigation

VLN task requires the agent to navigate in a 3D environment to a target location following language instructions. Our base network for performing this task is derived from sequence-to-sequence baseline described in [58]. Similar to Sec. V-A, we replace visual encoder with our Scene Graph Transformer. The network is trained end-to-end using teacher-forcing IL training [63], [58]. Teacher-forcing minimizes the maximum-likelihood loss using ground-truth trajectories. We again include an auxiliary reward equal to the change in the percentage of the ground truth scene graph observed by the agent.

Quantitative Results. In Table I, we compare several VLN baselines on VLN-CE dataset validation seen split [58]. The table is divided into No-Learning and Learning parts to aid our study. The compared models are: **1) No-Learning (i.e., Random and Hand-Crafted):** Agents do not process any sensor input and have no learned component [58]. **2) Learning (i.e., Sequence to Sequence):** We use Sequence to Sequence (Seq2Seq) VLN baseline following [58], [57]. Seq2Seq agent employs a recurrent policy that takes a representation of instructions and visual observations (RGB, Depth, GraphMapper) at each time step and then predicts an action.

The agents are trained following [58] for at most 30 epochs and the best performing models are selected. We use the Adam optimizer and a learning rate of $2.5e{-}4$. We use ResNet50 model [60] pretrained on ImageNet to extract RGB Image features and ResNet50 model pretrained on PointGoal task with DDPPO [23] to extract Depth features. The models are implemented in PyTorch [64], on top of Habitat-API [19] version 0.1.5 and trained utilizing four RTX 2080 Ti GPUs.

We provide Seq2Seq models with different inputs in Table I. We observe that the RGB model only performs on par with the no-learning baselines. The Seq2Seq Depth baseline shows a large improvement due to having access to ground-truth depth information. We note that, although such explicit dense depth cues help the agent significantly to traverse effectively, the performance of the agent is unlikely to transfer well to real-world settings due to the noisy and sparse nature of sensor data available from existing depth-sensing technologies. On the other hand, GraphMapper only depends on implicit depth (3D positions) and, hence, is more likely to transfer well. The GraphMapper agent performs significantly better than the RGB agent (*e.g.*, 0.18 *vs.* 0.03 in SR) and GraphMapper-Depth agent outperforms the RGB-Depth agent (*e.g.*, 0.27 *vs.* 0.25 in SR). Furthermore, GraphMapper-Depth yields the best results, which again shows that GraphMapper can act as a modular component to improve the performance of other models.

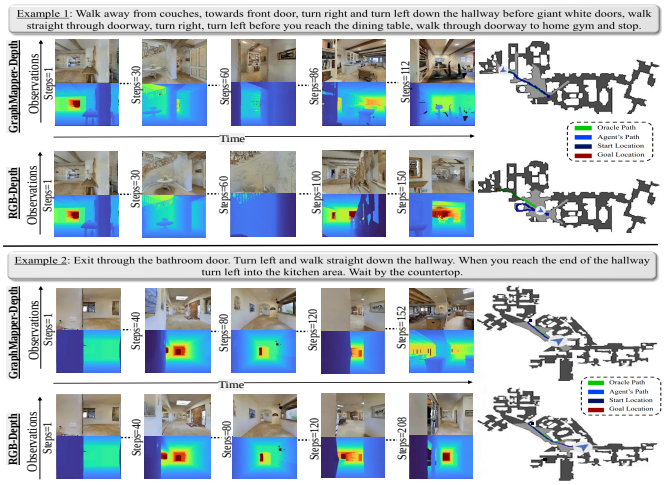

Fig. 4: Figure shows instruction-following trajectories of our GraphMapper-Depth agent compared to RGB-Depth in environments within VLN-CE. Sample observations (i.e., RGB, Depth) seen by the agent are shown at each timestep. The top-down map (shown on the right) is not available to the agent and is only used for evaluation.

Qualitative Results. In Figure 4, we qualitatively compare our GraphMapper-Depth agent to RGB-Depth agent. Both the examples represent success cases for our agent whereas RGB-Depth fails due to a lack of semantic understanding. In Example 1, we see RGB-Depth agent is unable to recognize the hallway before giant white doors, gets stuck near the staircase, and returns close to the start location. On the contrary, our agent successfully follows the complex sequence of instructions to finally perform STOP action after reaching its goal home gym. In Example 2, we see both the agents follow the instructions to reach close to the goal. The RGB-Depth agent reaches the kitchen but fails to identify the countertop and continues to take actions for several steps and incorrectly performs a STOP action in a hallway. Our GraphMapper-Depth agent is able to immediately understand that it is close to its goal location countertop after entering kitchen and performs a STOP action.

VI. Conclusion

In this work, we have introduced a novel method for improving visual feature representations in learning-based visual navigation systems, GraphMapper. We have shown that using structurally and semantically refined features produced from our Scene Graph Transformer Network increases the navigational success and sample efficiency of these models. Furthermore, we demonstrate that GraphMapper is capable of generating scene graph representations of its environment that can be useful for explaining black box learning policies or performing other downstream mapping tasks. In future work, we hope to explore the use of these scene representations for higher-level tasks, such as active SLAM or object finding, or for enabling better human interaction with autonomous systems.

REFERENCES

[1] B. Zhou, P. Krahenbuhl, and V. Koltun, "Does computer vision matter for action?" *arXiv preprint arXiv:1905.12887*, 2019.

[2] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[3] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," *NeurIPS*, pp. 11 983–11 993, 2019.

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

[5] A. J. Davison and D. W. Murray, "Mobile robot localisation using active vision," in *ECCV*, 1998, pp. 809–825.

[6] G. N. DeSouza and A. C. Kak, "Vision for mobile robot navigation: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 2, pp. 237–267, 2002.

[7] A. Krasner, M. Sizintsev, A. Rajvanshi, H.-P. Chiu, N. Mithun, K. Kaighn, P. Miller, R. Villamil, and S. Samarasekera, "Signav: Semantically-informed gps-denied navigation and mapping in visually-degraded environments," in *WACV*, 2022, pp. 2972–2981.

[8] S. M. LaValle, J. J. Kuffner, B. Donald *et al.*, "Rapidly-exploring random trees: Progress and prospects," *Algorithmic and computational robotics: new directions*, vol. 5, pp. 293–308, 2001.

[9] J. Canny, *The complexity of robot motion planning*. MIT press, 1988.

[10] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[11] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016.

[12] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *CVPR workshops*, 2017, pp. 16–17.

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

[14] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *ICML*, 2016, pp. 1928–1937.

[15] T. Chen, S. Gupta, and A. Gupta, "Learning exploration policies for navigation," in *ICLR*, 2019.

[16] E. Kolve, R. Mottaghi, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi, "Ai2-thor: An interactive 3d environment for visual ai," *arXiv preprint arXiv:1712.05474*, 2017.

[17] A. Dosovitskiy and V. Koltun, "Learning to act by predicting the future," *arXiv preprint arXiv:1611.01779*, 2016.

[18] D. S. Chaplot, D. Gandhi, S. Gupta, A. Gupta, and R. Salakhutdinov, "Learning to explore using active neural SLAM," in *ICLR*, 2020.

[19] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: A Platforotion for Embodied AI Research," in *ICCV*, 2019.

[20] D. Mishkin, A. Dosovitskiy, and V. Koltun, "Benchmarking classic and learned navigation in complex 3d environments," *arXiv preprint arXiv:1901.10915*, 2019.

[21] A. Sax, B. Emi, A. R. Zamir, L. J. Guibas, S. Savarese, and J. Malik, "Mid-level visual representations improve generalization and sample efficiency for learning active tasks," *arXiv preprint arXiv:1812.11971*, 2018.

[22] W. B. Shen, D. Xu, Y. Zhu, L. J. Guibas, F. Li, and S. Savarese, "Situational fusion of visual representation for visual navigation," in *ICCV*, 2019.

[23] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra, "Decentralized distributed ppo: Solving pointgoal navigation," *arXiv preprint arXiv:1911.00357*, 2019.

[24] S. K. Ramakrishnan, Z. Al-Halah, and K. Grauman, "Occupancy anticipation for efficient exploration and navigation," in *ECCV*. Springer, 2020, pp. 400–418.

[25] D. S. Chaplot, D. P. Gandhi, A. Gupta, and R. R. Salakhutdinov, "Object goal navigation using goal-oriented semantic exploration," *NeurIPS*, 2020.

[26] N. Savinov, A. Raichuk, R. Marinier, D. Vincent, M. Pollefeys, T. Lillicrap, and S. Gelly, "Episodic curiosity through reachability," *arXiv preprint arXiv:1810.02274*, 2018.

[27] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," in *CVPR*, 2017, pp. 7272–7281.

[28] V. Cartillier, Z. Ren, N. Jain, S. Lee, I. Essa, and D. Batra, "Semantic mapnet: Building allocentric semanticmaps and representations from egocentric views," *arXiv preprint arXiv:2010.01191*, 2020.

[29] B. Yamauchi, "A frontier-based approach for autonomous exploration." in *cira*, vol. 97, 1997, p. 146.

[30] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte *et al.*, "Minerva: A second-generation museum tour-guide robot," in *ICRA*, vol. 3. IEEE, 1999.

[31] Z. Seymour, K. Thopalli, N. C. Mithun, H.-P. Chiu, S. Samarasekera, and R. Kumar, "Maast: Map attention with semantic transformers for efficient visual navigation," in *ICRA*, 2021.

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

[33] E. Beeching, J. Dibangoye, O. Simonin, and C. Wolf, "Learning to plan with uncertain topological maps," in *ECCV*. Springer, 2020, pp. 473–490.

[34] H. Wang, W. Wang, W. Liang, C. Xiong, and J. Shen, "Structured scene memory for vision-language navigation," in *CVPR*, 2021, pp. 8455–8464.

[35] D. S. Chaplot, R. Salakhutdinov, A. Gupta, and S. Gupta, "Neural topological slam for visual navigation," in *CVPR*, 2020, pp. 12 875–12 884.

[36] Y. Li, W. Ouyang, B. Zhou, K. Wang, and X. Wang, "Scene graph generation from objects, phrases and region captions," in *ICCV*, 2017, pp. 1270–1279.

[37] J. Johnson, A. Gupta, and L. Fei-Fei, "Image generation from scene graphs," in *CVPR*, 2018, pp. 1219–1228.

[38] K. Mo, P. Guerrero, L. Yi, H. Su, P. Wonka, N. Mitra, and L. J. Guibas, "Structurenet: Hierarchical graph networks for 3d shape generation," *arXiv preprint arXiv:1908.00575*, 2019.

[39] O. Ashual and L. Wolf, "Specifying object attributes and relations in interactive scene generation," in *ICCV*, 2019, pp. 4561–4569.

[40] S. Qi, Y. Zhu, S. Huang, C. Jiang, and S.-C. Zhu, "Human-centric indoor scene synthesis using stochastic grammar," in *CVPR*, 2018, pp. 5899–5908.

[41] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[42] C. Desai, D. Ramanan, and C. C. Fowlkes, "Discriminative models for static human-object interactions," in *CVPR Workshops*, 2010.

[43] C. Lu, R. Krishna, M. Bernstein, and L. Fei-Fei, "Visual relationship detection with language priors," in *ECCV*. Springer, 2016, pp. 852–869.

[44] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei, "Scene graph generation by iterative message passing," in *CVPR*, 2017, pp. 5410–5419.

[45] J. Yang, J. Lu, S. Lee, D. Batra, and D. Parikh, "Graph r-cnn for scene graph generation," in *ECCV*, 2018, pp. 670–685.

[46] Y. Li, W. Ouyang, B. Zhou, J. Shi, C. Zhang, and X. Wang, "Factorizable net: An efficient subgraph-based framework for scene graph generation," in *ECCV*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., 2018, pp. 346–363.

[47] Y. Zhou, Z. While, and E. Kalogerakis, "Scenegraphnet: Neural message passing for 3d indoor scene augmentation," in *ICCV*, 2019, pp. 7384–7392.

[48] I. Armeni, Z. He, J. Gwak, A. R. Zamir, M. Fischer, J. Malik, and S. Savarese, "3d scene graph: A structure for unified semantics, 3d space, and camera," in *ICCV*, 2019.

[49] P. Gay, J. Stuart, and A. Del Bue, "Visual graphs from motion (vgfm): Scene understanding with object geometry reasoning," in *ACCV*, 2018.

[50] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *ICCV*, 2017, pp. 2961–2969.

[51] F. Xia, A. R. Zamir, Z.-Y. He, A. Sax, J. Malik, and S. Savarese, "Gibson Env: Real-world perception for embodied agents," in *CVPR*, 2018, pp. 9068–9079.

[52] M. Savva, A. X. Chang, A. Dosovitskiy, T. A. Funkhouser, and V. Koltun, "Minos: Multimodal indoor simulator for navigation in complex environments," *arXiv preprint arXiv:1712.03931*, 2017.

[53] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L. Li, D. A. Shamma, M. S. Bernstein, and L. Fei-Fei, "Visual genome: Connecting language and vision using crowdsourced dense image annotations," *International Journal of Computer Vision*, vol. 123, no. 1, pp. 32–73, 2017.

[54] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," in *NeurIPS*, 2015.

[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.

[56] P. Anderson, A. X. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, and A. R. Zamir, "On evaluation of embodied navigation agents," *arXiv preprint arXiv:1807.06757*, 2018.

[57] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. van den Hengel, "Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments," in *CVPR*, 2018.

[58] J. Krantz, E. Wijmans, A. Majumdar, D. Batra, and S. Lee, "Beyond the nav-graph: Vision-and-language navigation in continuous environments," in *ECCV*, 2020, pp. 104–120.

[59] G. I. Magalhaes, V. Jain, A. Ku, E. Ie, and J. Baldridge, "General evaluation for instruction conditioned navigation using dynamic time warping," in *NeurIPS ViGIL Workshop*, 2019.

[60] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[61] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *EMNLP*, 2014, pp. 1724–1734.

[62] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[63] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[64] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *NeurIPS*, vol. 32, pp. 8026–8037, 2019.