BAYESIAN STATE TRACKING AND SIM-TO-REAL TRANSFER FOR VISION-AND-LANGUAGE NAVIGATION

A Thesis
Presented to
The Academic Faculty

By

Ayush Shrivastava

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science in the
College of Computing

Georgia Institute of Technology

May 2021

BAYESIAN STATE TRACKING AND SIM-TO-REAL TRANSFER FOR VISION-AND-LANGUAGE NAVIGATION

To my parents, Shruti di and Stuti di, for always being there for me.

To Devi and Dhruv, for believing in me.

ACKNOWLEDGMENTS

I have spent three wonderful years at Georgia Tech, the first year as an intern and the other two as a Masters student. There are several people that I must thank for making this journey delightful.

I would first like to thank my advisor, Devi Parikh for guiding me and funding me throughout the course of my Masters. Devi has been constant source of support, guidance and encouragement over the course of years. I have always felt comfortable reaching out to her regarding questions, both work-related and personal, and she has always been helpful in providing insights, no matter how stupid the question may sound. I have been fascinated by her time-management and organizational skills which I sincerely hope to fully adopt someday. Her clarity of thought and probing questions have helped me decide between seemingly difficult choices. I have always enjoyed her enthusiasm for our dinner-table questions which have at times forced to think critically on topics outside work.

I must also thank Dhruv Batra, who I have had the privilege of working closely with. I have been lucky to have, not one, but two advisors, Dhruv and Devi. Right after my undergrad, Dhruv had hired me as an intern in the lab which became one of the best opportunities for my career so far. Thank you for having taken a chance with me! His honest feedback has helped shape my thought process and attitude towards research. I immensely enjoyed his Deep Learning class at Georgia Tech which I plan to revisit whenever I want to revise these concepts.

During my time at Georgia Tech, I have been fortunate enough to work with Peter Anderson on several projects. Peter had been quite generous with his time and taught me so much while working with me. Showing how to do research, handholding me through designing experiments, how to interpret the results that we get and re-explaining concepts and ideas that I could not pick up the first time, he was involved in every major step of the project and constantly motivated me to do high-quality work.

I have also had the pleasure of working with Stefan Lee on these projects. He was a constant source of motivation and helped me take a step back to see the high-level ideas of the research projects. I am grateful for his help in presenting our NeurIPS poster which was my first time presenting at a conference and the feedback he gave for the presentation was super helpful. Thank you for always making me feel welcome for boardgame sessions at your home and lab retreats.

During the summers 2020, I was fortunate to intern at Amazon Alexa AI with Karthik Gopalakrishnan and Dilek Hakkani-Tür. Both of them helped me immensely in navigating my internship and were very supportive throughout the course of my internship and for our continued collaboration after the summers as well. I would like to specially thank Karthik for the discussions about the low-level details of the project and for coming up with ideas for us to try next.

Special thanks to Arjun Majumdar, who I collaborated with on our ECCV paper. Working with him was a learning experience for me and I learned a lot from our brainstorming discussions about the project. Also, thanks to Aishwarya Agrawal and Yash Goyal for all the help with the VQA challenge and the VQA workshop, and for teaching me what all goes in organizing a workshop.

I would also like to thank all members of the CVMLP lab with whom I had a chance to interact with: Rishabh Jain, Harsh Agrawal, Ramprasaath Selvaraju, Prithvijit Chattopadhyay, Deshraj Yadav, Abhishek Das, Karan Desai, Vishvak Murahari, Samyak Datta, Purva Tendulkar, Viraj Prabhu, Nirbhay Modhe, Jiasen Lu, Jianwei Yang, Arjun Chandrasekaran, Ramakrishna Vedantam, Sashank Gondala, Abhinav Moudgil, Yash Kant, Erik Wijmans, Joanne Truong, Joel Ye, Sameer Dharur, Zhile Ren, Ashwin Kalyan, Michael Cogswell, Yash Goyal, Aishwarya Agrawal, Sanyam Agrawal. I owe special thanks to my flatmates, Rishabh and Harsh for bearing me for 2 years, letting me install tech gadgets all around the house and teaching me new recipes to cook. Special thanks to Harsh, Prithvi and Deshraj for guiding me throughout my grad school and helping with my queries just about anything.

Thanks to Rishabh for all the expedited help related to EvalAI. Thanks to Prithvi, Ram, Rishabh and Harsh for making this pandemic a little bit more tolerable and for our fun weekend hangouts. Thanks to Karan, Rishabh, Deshraj, Harsh and Prithvi – for making my first year in the US enjoyable. Thanks to Rishabh, Harsh, Prithvi, Abhishek, Vishvak – for our fun little 8-ball pool breaks in CODA.

Finally, I am grateful to my parents, Shruti di and Stuti di who provided constant love, encouragement and support to pursue my dreams and made significant sacrifices to enable my move to the United States to pursue graduate school. This accomplishment would not have been possible without them. And to Anandita, for always being there for me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

A visually-grounded navigation instruction can be interpreted as a sequence of expected observations and actions an agent following the correct trajectory would encounter and perform. Based on this intuition, we formulate the problem of finding the goal location in Vision-and-Language Navigation (VLN) [1] within the framework of Bayesian state tracking – learning observation and motion models conditioned on these expectable events. Together with a mapper that constructs a semantic spatial map on-the-fly during navigation, we formulate an end-to-end differentiable Bayes filter and train it to identify the goal by predicting the most likely trajectory through the map according to the instructions. The resulting navigation policy constitutes a new approach to instruction following that explicitly models a probability distribution over states, encoding strong geometric and algorithmic priors while enabling greater explainability. Our experiments show that our approach outperforms a strong LingUNet [2] baseline when predicting the goal location on the map. On the full VLN task, i.e., navigating to the goal location, our approach achieves promising results with less reliance on navigation constraints.

In the second half of the thesis, we study the challenging problem of releasing a robot in a previously unseen environment, and having it follow unconstrained natural language navigation instructions. Recent work on the task of VLN has achieved significant progress in simulation. To assess the implications of this work for robotics, we transfer a VLN agent trained in simulation to a physical robot. To bridge the gap between the high-level discrete action space learned by the VLN agent, and the robot's low-level continuous action space, we propose a subgoal model to identify nearby waypoints, and use domain randomization to mitigate visual domain differences. For accurate sim and real comparisons in parallel environments, we annotate a 325m^2 office space with 1.3km of navigation instructions, and create a digitized replica in simulation. We find that sim-to-real transfer to an environment not seen in training is successful if an occupancy map and navigation graph can be collected

and annotated in advance (success rate of 46.8% vs. 55.9% in sim), but much more challenging in the hardest setting with no prior mapping at all (success rate of 22.5%).

CHAPTER 1
VISION-AND-LANGUAGE NAVIGATION AS BAYESIAN STATE TRACKING

1.1 Introduction

One long-term challenge in AI is to build agents that can navigate complex 3D environments from natural language instructions. In the Vision-and-Language Navigation (VLN) instantiation of this task [1], an agent is placed in a photo-realistic reconstruction of an indoor environment and given a natural language navigation instruction, similar to the example in Figure 1.1. The agent must interpret this instruction and execute a sequence of actions to navigate efficiently from its starting point to the corresponding goal. This task is challenging for existing models [3, 4, 5, 6, 7, 8, 9], particularly as the test environments are unseen during training and no prior exploration is permitted in the hardest setting.

To be successful, agents must learn to ground language instructions to both visual observations and actions. Since the environment is only partially-observable, this in turn requires the agent to relate instructions, visual observations and actions through memory. Current approaches to the VLN task use unstructured general purpose memory representations implemented with recurrent neural network (RNN) hidden state vectors [1, 3, 4, 5, 6, 7, 8, 9]. However, these approaches lack geometric priors and contain no mechanism for reasoning about the likelihood of alternative trajectories – a crucial skill for the task, e.g., 'Would this look more like the goal if I was on the other side of the room?'. Due to this limitation, many previous works have resorted to performing inefficient first-person search through the environment using search algorithms such as beam search [7, 5]. While this greatly improves performance, it is clearly inconsistent with practical applications like robotics since the resulting agent trajectories are enormously long – in the range of hundreds or thousands of meters.

To address these limitations, it is essential to move towards reasoning about alternative trajectories in a *representation* of the environment – where there are no search costs associated with moving a physical robot – rather than in the environment itself. Towards this, we extend the Matterport3D simulator [1] to provide depth outputs, enabling us to investigate the use of a *semantic spatial map* [10, 11, 12, 13] in the context of the VLN task for the first time. We propose an instruction-following agent incorporating three components: (1) a *mapper* that builds a semantic spatial map of its environment from first-person views; (2) a *filter* that determines the most probable trajectory(ies) and goal location(s) in the map, and (3) a *policy* that executes a sequence of actions to reach the predicted goal.

From a modeling perspective, our key contribution is the filter that formulates instruction following as a problem of Bayesian state tracking [14]. We notice that a visually-grounded navigation instruction typically contains a description of expected future observations and actions on the path to the goal. For example, consider the instruction 'walk out of the bathroom, turn left, and go on to the bottom of the stairs and wait near the coat rack' shown in Figure 1.1. When following this instruction, we would expect to immediately observe a bathroom, and at the end a coat rack near a stairwell. Further, in reaching the goal we can anticipate performing certain actions, such as turning left and continuing that way. Based on this intuition, we use a sequence-to-sequence model with attention to extract sequences of latent vectors representing observations and actions from a natural language instruction.

Faced with a known starting state, a (partially-observed) semantic spatial map generated

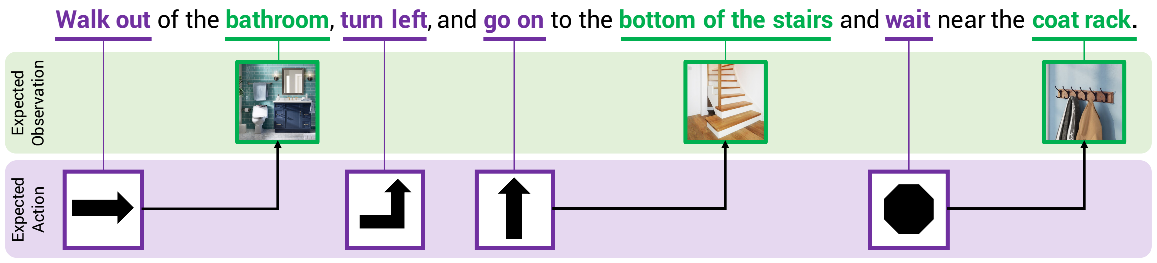

Walk out of the bathroom, turn left, and go on to the bottom of the stairs and wait near the coat rack.
Expected Observation
Expected Action

Figure 1.1: Navigation instructions can be interpreted as encoding a set of latent expectable observations and actions an agent would encounter and undertake while successfully following the directions.

by the mapper, and a sequence of (latent) observations and actions, we now quite naturally interpret our instruction following task within the framework of Bayesian state tracking. Specifically, we formulate an end-to-end differentiable histogram filter [15] with learnable observation and motion models, and we train it to predict the most likely trajectory taken by a human demonstrator. We emphasize that we are not tracking the state of the *actual agent*. In the VLN setting, the pose of the agent is known with certainty at all times. The key challenge lies in determining the location of the natural-language-specified goal state. Leveraging the machinery of Bayesian state estimation allows us to reason in a principled fashion about what a (hallucinated) human demonstrator would do when following this instruction – by explicitly modeling the demonstrator's trajectory over multiple time steps in terms of a probability distribution over map cells. The resulting model encodes both strong geometric priors (e.g., pinhole camera projection) and strong algorithmic priors (e.g., explicit handling of uncertainty, which can be multi-modal), while enabling explainability of the learned model. For example, we can separately examine the motion model, the observation model, and their interaction during filtering.

Empirically, we show that our filter-based approach significantly outperforms a strong LingUNet [2] baseline when tasked with predicting the goal location in VLN given a partially-observed semantic spatial map. On the full VLN task (incorporating the learned policy as well), our approach achieves a success rate on the test server [1] of 32.7% (29.9% `SPL` [16]), a credible result for a new class of model trained exclusively with imitation learning and without data augmentation. Although our policy network is specific to the Matterport3D simulator environment, the rest of our pipeline is general and operates without knowledge of the simulator's navigation graph (which has been heavily utilized in previous work [1, 3, 4, 5, 6, 7, 8, 9]). We anticipate this could be an advantage for sim-to-real transfer (i.e., in real robot scenarios where a navigation graph is not provided, and could be non-trivial to generate).

Contributions. In summary, we:

- Extend the existing Matterport3D simulator [1] used for VLN to support depth image outputs.
- Implement and investigate a semantic spatial memory in the context of VLN for the first time.
- Propose a novel formulation of instruction following / goal prediction as Bayesian state tracking of a hypothetical human demonstrator.
- Show that our approach outperforms a strong baseline for goal location prediction.
- Demonstrate credible results on the full VLN task with the addition of a simple reactive policy, with less reliance on navigation constraints than prior work.

1.2 Related work

Vision-and-Language Navigation Task. The VLN task [1], based on the Matterport3D dataset [17], builds on a rich history of prior work on situated instruction-following tasks beginning with SHRDLU [18]. Despite the task's difficulty, a recent flurry of work has seen significant improvements in success rates and related metrics [3, 4, 5, 6, 7, 8, 9]. Key developments include the use of instruction-generation ('speaker') models for trajectory re-ranking and data augmentation [7, 8], which have been widely adopted. Other work has focused on developing modules for estimating progress towards the goal [5] and learning when to backtrack [6, 9]. However, comparatively little attention has been paid to the memory architecture of the agent. LSTM [19] memory has been used in all previous work.

Memory architectures for navigation agents. Beyond the VLN task, various categories of memory structures for deep neural navigation agents can be identified in the literature, including unstructured, addressable, metric and topological. General purpose *unstructured* memory representations, such as LSTM memory [19], have been used extensively in both 2D and 3D environments [20, 21, 22, 23, 24]. However, LSTM memory does not offer context-dependent storage or retrieval, and so does not naturally facilitate local reasoning when navigating large or complex environments [25]. To overcome these limitations, both *addressable* [25, 26] and *topological* [27] memory representations have been proposed for navigating in mazes and for predicting free space. However, in this work we elect to use a *metric* semantic spatial map [10, 11, 12, 13] – which preserves the geometry of the environment – as our agent's memory representation since reasoning about observed phenomena from alternative viewpoints is an important aspect of the VLN task. Semantic spatial maps are grid-based representations containing convolutional neural network (CNN) features which have been recently proposed in the context of visual navigation [10], interactive question answering [13], and localization [12]. However, there has been little work on incorporating these memory representations into tasks involving natural language. The closest work to ours is [11], however our map construction is more sophisticated as we use depth images and do not assume that all pixels lie on the ground plane. Furthermore, our major contribution is formulating instruction-following as Bayesian state tracking.

1.3 Preliminaries: Bayes filters

A Bayes filter [14] is a framework for estimating a probability distribution over a latent state $\boldsymbol{s}$ (e.g., the pose of a robot) given a history of observations $\boldsymbol{o}$ and actions $\boldsymbol{a}$ (e.g., camera observations, odometry, etc.). At each time step t the algorithm computes a posterior probability distribution $bel(\boldsymbol{s}_t) = p(\boldsymbol{s}_t \mid \boldsymbol{a}_{1:t}, \boldsymbol{o}_{1:t})$ conditioned on the available data. This is also called the *belief*.

Taking as a key assumption the Markov property of states, and conditional independence between observations and actions given the state, the belief $bel(\boldsymbol{s}_t)$ can be recursively updated from $bel(\boldsymbol{s}_{t-1})$ using two alternating steps to efficiently combine the available evidence. These steps may be referred to as the *prediction* based on action $\boldsymbol{a}_t$ and the *observation update* using observation $\boldsymbol{o}_t$.

Prediction. In the prediction step, the filter processes the action $\boldsymbol{a}_t$ using a *motion model*

$p(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}, \boldsymbol{a}_t)$ that defines the probability of a state $\boldsymbol{s}_t$ given the previous state $\boldsymbol{s}_{t-1}$ and an action $\boldsymbol{a}_t$. In particular, the updated belief $\overline{bel}(\boldsymbol{s}_t)$ is obtained by integrating (summing) over all prior states $\boldsymbol{s}_{t-1}$ from which action $\boldsymbol{a}_t$ could have lead to $\boldsymbol{s}_t$, as follows:

$$\overline{bel}(\boldsymbol{s}_t) = \int p(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}, \boldsymbol{a}_t)\; bel(\boldsymbol{s}_{t-1})\; d\boldsymbol{s}_{t-1} \tag{1.1}$$

Observation update. During the observation update, the filter incorporates information from the observation $\boldsymbol{o}_t$ using an *observation model* $p(\boldsymbol{o}_t \mid \boldsymbol{s}_t)$ which defines the likelihood of an observation $\boldsymbol{o}_t$ given a state $\boldsymbol{s}_t$. The observation update is given by:

$$bel(\boldsymbol{s}_t) = \eta\; p(\boldsymbol{o}_t \mid \boldsymbol{s}_t)\; \overline{bel}(\boldsymbol{s}_t) \tag{1.2}$$

where η is a normalization constant and Equation 1.2 is derived from Bayes rule.

Differentiable implementations. To apply Bayes filters in practice, a major challenge is to construct accurate probabilistic motion and observation models for a given choice of belief representation $bel(\boldsymbol{s}_t)$. However, recent work has demonstrated that Bayes filter implementations – including Kalman filters [28], histogram filters [15] and particle filters [29, 30] – can be embedded into deep neural networks. The resulting models may be seen as new recurrent architectures that encode algorithmic priors from Bayes filters (e.g., explicit representations of uncertainty, conditionally independent observation and motion models) yet are fully differentiable and end-to-end learnable.

1.4 Agent model

In this section, we describe our VLN agent that simultaneously: (1) builds a semantic spatial map from first-person views; (2) determines the most probable goal location in the current map by filtering likely trajectories taken by a human demonstrator from the start location (i.e., the 'ghost'); and (3) executes actions to reach the predicted goal. Each of these functions is the responsibility of a separate module which we refer to as the *mapper*, *filter*, and *policy*, respectively. We begin with the mapper.

1.4.1 Mapper

At each time step t, the mapper updates a learned semantic spatial map $\mathcal{M}_t \in \mathbb{R}^{M \times Y \times X}$ in the world coordinate frame from first-person views. This map is a grid-based metric representation in which each grid cell contains a M-sized latent vector representing the visual appearance of a small corresponding region in the environment. X and Y are the spatial dimensions of the semantic map, which could be dynamically resized if necessary. The map maintains a representation for every world coordinate (x, y) that has been observed by the agent, and each map cell is computed from all past observations of the region. We define the world coordinate frame by placing the agent at the center of the map at the start of each episode, and defining the xy plane to coincide with the ground plane.

Inputs. As with previous work on VLN task [7, 5, 6], we provide the agent with a panoramic view of its environment at each time step[1] comprised of a set of RGB images $\mathcal{I}_t = \{I_{t,1}, I_{t,2}, \ldots, I_{t,K}\}$, where $I_{t,k}$ represents the image captured in direction k. The agent also receives the associated depth images $\mathcal{D}_t = \{D_{t,1}, D_{t,2}, \ldots, D_{t,K}\}$ and camera poses $\mathcal{P}_t = \{P_{t,1}, P_{t,2}, \ldots, P_{t,K}\}$. We additionally assume that the camera intrinsics and the ground plane are known. In the VLN task, these inputs are provided by the simulator, in other settings they could be provided by SLAM systems etc.

Image processing. Each image $I \in \mathbb{R}^{H \times W \times 3}$ is processed with a pretrained convolutional neural network (CNN) to extract a downsized visual feature representation $\boldsymbol{v} \in \mathbb{R}^{H' \times W' \times C}$. To extract a corresponding depth image $\boldsymbol{d} \in \mathbb{R}^{H' \times W'}$, we apply 2D adaptive average pooling to the original depth image $D \in \mathbb{R}^{H \times W}$. Missing (zero) depth values are excluded from the pooling operation.

Feature projection. Similarly to MapNet [12], we project CNN features $\boldsymbol{v}$ onto the ground

[1]The panoramic setting is chosen for comparison with prior work – not as a requirement of our architecture.

plane in the world coordinate frame using the corresponding depth image $\boldsymbol{d}$, the camera pose P, and a pinhole camera model using known camera intrinsics. We then discretize the projected features into a 2D spatial grid $\mathcal{F}_t \in \mathbb{R}^{C \times Y \times X}$, using elementwise max pooling to handle feature collisions in a cell.

Map update. To integrate map observations $\mathcal{F}_t$ into our semantic spatial map $\mathcal{M}_t$, we use a convolutional implementation [31] of a Gated Recurrent Unit (GRU) [32]. In preliminary experiments we found that using convolutions in both the input-to-state and state-to-state transitions reduced the variance in the performance of the complete agent by sharing information across neighboring map cells. However, since both the map $\mathcal{M}_t$ and the map update $\mathcal{F}_t$ are sparse, we use a sparsity-aware convolution operation that evaluates only observed pixels and normalizes the output [33]. We also mask the GRU map update to prevent bias terms from accumulating in the unobserved regions.

1.4.2 Filter

At the beginning of each episode the agent is placed at a start location $\boldsymbol{s}_0^* = (x_0, y_0, \theta_0)$, where θ represents the agent's heading and x and y are coordinates in the world frame as previously described. The agent is given an instruction $\mathcal{X}$ describing the trajectory to an unknown goal coordinate $\boldsymbol{s}_T^* = (x_T, y_T, \cdot)$. As an intermediate step towards actually reaching the goal, we wish to identify likely goal locations in the partially-observed semantic spatial map $\mathcal{M}$ generated by the mapper.

Our approach to this problem is based on the observation that a natural language navigation instruction typically conveys a sequence of expected future observations and actions, as previously discussed. Based on this observation, we frame the problem of determining the goal location $\boldsymbol{s}_T^*$ as a tracking problem. As illustrated in Figure 1.2 and described further below, we implement a Bayes filter to track the pose $\boldsymbol{s}_t^*$ of a hypothetical human demonstrator (i.e., the 'ghost') from the start location to the goal. As inputs to the filter, we provided a series of latent observations $\boldsymbol{o}_t$ and actions $\boldsymbol{a}_t$ extracted from the

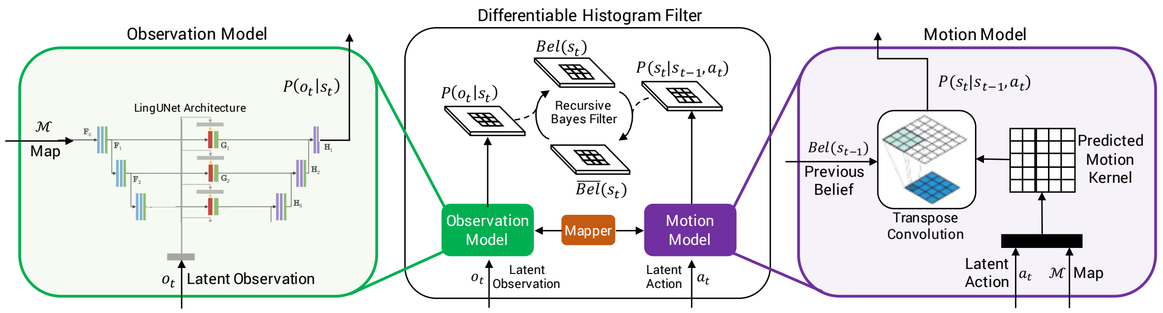


Figure 1.2: Proposed filter architecture. To identify likely goal locations in the partially-observed semantic spatial map $\mathcal{M}$ generated by the mapper, we first initialize the belief $bel(\boldsymbol{s}_t)$ with the known starting state $\boldsymbol{s}_0$. We then recursively: (1) generate a latent observation $\boldsymbol{o}_t$ and action $\boldsymbol{a}_t$ from the instruction, (2) compute the prediction step using the motion model (Equation 1.3), and (3) compute the observation update using the observation model (Equation 1.5), stopping after T time steps. The resulting belief $bel(\boldsymbol{s}_T)$ represents the posterior probability distribution over likely goal locations.

navigation instruction $\mathcal{X}$. The output of the filter is the belief over likely goal locations $bel(\boldsymbol{s}_T)$.

Note that in this section we use the subscript t to denote time steps in the filter, overloading the notation from subsection 1.4.1 in which t referred to agent time steps. We wish to make clear that in our model the filter runs in an inner loop, re-estimating belief over trajectories taken by a demonstrator starting from $\boldsymbol{s}_0$ each time the map is updated by the agent in the outer loop.

Belief. We define the state $\boldsymbol{s}_t = (x_t, y_t, \theta_t)$ using the agent's (x, y) position and heading θ. We represent the belief over the demonstrator's state at each time step t with a histogram, implemented as a tensor $bel(\boldsymbol{s}_t) = \boldsymbol{b}_t$, $\boldsymbol{b}_t \in \mathbb{R}^{\Theta \times Y \times X}$ where X, Y and Θ are the number of bins for each component of the state, respectively. Using a histogram-based approach allows the filter to track multiple hypotheses, meshes easily with our implementation of a grid-based semantic map, and leads naturally to an efficient motion model implementation based on convolutions, as discussed further below. However, our proposed approach could also be implemented as a particle filter [29, 30], for example if discretization error was a significant concern.

Observations and actions. To transform the instruction $\mathcal{X}$ into a latent representation of observations $\boldsymbol{o}$ and actions $\boldsymbol{a}$, we use a sequence-to-sequence model with attention [34]. We first tokenize the instruction into a sequence of words $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_l\}$ which are encoded using learned word embeddings and a bi-directional LSTM [19] to output a series of encoder hidden states $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_l\}$ and a final hidden state $\boldsymbol{e}$ representing the output of a complete pass in each direction. We then use an LSTM decoder to generate a series of latent observation and action vectors $\{\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_T\}$ and $\{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_T\}$ respectively. Here, $\boldsymbol{o}_t$ is given $\boldsymbol{o}_t = [\hat{\boldsymbol{e}}_t^o, \boldsymbol{h}_t]$, where $\boldsymbol{h}_t$ is the hidden state of the decoder LSTM, and $\hat{\boldsymbol{e}}_t^o$ is the attended instruction representation computed using a standard dot-product attention mechanism [35]. The action vectors $\boldsymbol{a}_t$ are computed analogously, using the same decoder LSTM but with a separate learned attention mechanism. The only input to the decoder LSTM is a positional encoding [36] of the decoding time step t. While the correct number of decoding time steps T is unknown, in practice we always run the filter for a fixed number of time steps equal to the maximum trajectory length in the dataset (which is 6 steps in the navigation graph).

Motion model. We implement the *motion model* $p(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}, \boldsymbol{a}_t, \mathcal{M})$ as a convolution over the belief $\boldsymbol{b}_{t-1}$. This ensures that agent motion is consistent across the state space while explicitly enforcing locality, i.e., the agent cannot move further than half the kernel size in a single time step. Similarly to [15], the prediction step from Equation 1.1 is thus reformulated as:

$$\overline{\boldsymbol{b}}_t = \boldsymbol{b}_{t-1} * g(\boldsymbol{a}_t, \mathcal{M}) \tag{1.3}$$

where we define an action- and map-dependent motion kernel $g(\boldsymbol{a}_t, \mathcal{M}) \in \mathbb{R}^{\Theta^2 \times M^2}$ given by:

$$g(\boldsymbol{a}_t, \mathcal{M}) = \text{softmax}(\text{conv}([\boldsymbol{a}_t, \mathcal{M}])) \tag{1.4}$$

where conv is a small 3-layer CNN with ReLU activations operating on the semantic spatial map $\mathcal{M}$ and the spatially-tiled action vector $\boldsymbol{a}_t$, M is the motion kernel size and the softmax function enforces the prior that $g(\boldsymbol{a}_t, \mathcal{M})$ represents a probability mass function. Note that we include $\mathcal{M}$ in the input so that the motion model can learn that the agent is unlikely to move through obstacles.

Observation model. We require an *observation model* $p(\boldsymbol{o}_t \mid \boldsymbol{s}_t, \mathcal{M})$ to define the likelihood of a latent observation $\boldsymbol{o}_t$ conditioned on the agent's state $\boldsymbol{s}_t$ and the map $\mathcal{M}$. A generative observation model like this would be hard to learn, since it is not clear how to generate high-dimensional latent observations and normalization needs to be done across observations, not states. Therefore, we follow prior work [30] and learn a discriminative observation model that takes $\boldsymbol{o}_t$ and $\mathcal{M}$ as inputs and directly outputs the likelihood of this observation for each state. As detailed further in subsection 1.4.4, this observation model is trained end-to-end without direct supervision of the likelihood.

To implement our observation model we use LingUNet [2], a language-conditioned image-to-image network based on U-Net [37]. Specifically, we use the LingUNet implementation from Blukis et al. [11] with 3 cascaded convolution and deconvolution operations. The spatial dimensionality of the LingUNet output matches the input image (in this case, $\mathcal{M}$), and number of output channels is selected to match the number of heading bins Θ. Outputs are restricted to the range $[0, 1]$ using a sigmoid function. The observation update from Equation 1.2 is re-defined as:

$$\boldsymbol{b}_t = \eta \, \overline{\boldsymbol{b}}_t \odot \text{LingUNet}(\boldsymbol{o}_t, \mathcal{M}) \tag{1.5}$$

where η is a normalization constant and $\odot$ represents element-wise multiplication.

Goal prediction. In summary, to identify goal locations in the partially-observed spatial map $\mathcal{M}$, we initialize the belief $\boldsymbol{b}_0$ with the known starting state $\boldsymbol{s}_0$. We then iteratively: (1) Generate a latent observation $\boldsymbol{o}_t$ and action $\boldsymbol{a}_t$, (2) Compute the prediction step using

Equation 1.3, and (3) Compute the observation update using Equation 1.5. We stop after T filter update time steps. The resulting belief $\boldsymbol{b}_T$ represents the posterior probability distribution over goal locations.

1.4.3 Policy

The final component of our agent is a simple reactive policy network. It operates over a global action space defined by the complete set of panoramic viewpoints observed in the current episode (including both visited viewpoints, and their immediate neighbors). Our agent thus memorizes the local structure of the observed navigation graph to enable it to return to any previously observed location in a single action. The probability distribution over actions is defined by a softmax function, where the logit associated with each viewpoint i is given by $y_i = \text{MLP}([\boldsymbol{b}_{1:T,i}, \boldsymbol{v}_i])$, where MLP is a two-layer neural network, $\boldsymbol{b}_{1:T,i}$ is a vector containing the belief at each time step $1:T$ in a gaussian neighborhood around viewpoint i, and $\boldsymbol{v}_i$ is a vector containing the distance from the agent's current location to viewpoint i, and an indicator variable for whether i has been previously visited. If the policy chooses to revisit a previously visited viewpoint, we interpret this as a `stop` action. Note that our policy does not have direct access to any representation of the instruction, or the semantic map $\mathcal{M}$. Although our policy network is specific to the Matterport3D simulator environment, the rest of our pipeline is general and operates without knowledge of the simulator's navigation graph.

1.4.4 Learning

Our entire agent model is fully differentiable, from policy actions back to image pixels via the semantic spatial map, geometric feature projection function, etc. Training data for the model consists of instruction-trajectory pairs $(\mathcal{X}, \boldsymbol{s}^*_{1:T})$. In all experiments we train the filter using supervised learning by minimizing the KL-divergence between the predicted belief $\boldsymbol{b}_{1:T}$ and the true trajectory from the start to the goal $\boldsymbol{s}^*_{1:T}$, backpropagating gradients through the previous belief $\boldsymbol{b}_{t-1}$ at each step. Note that the predicted belief $\boldsymbol{b}_{1:T}$ is independent of the agent's actual trajectory $\boldsymbol{s}_{1:T}$ given the map $\mathcal{M}$. In the goal prediction experiments (subsection 1.5.2), the model is trained without a policy and so the agent's trajectory $\boldsymbol{s}_{1:T}$ is generated by moving towards the goal with 50% probability, or randomly otherwise. In the full VLN experiments (subsection 1.5.3), we train the filter concurrently with the policy. The policy is trained with cross-entropy loss to maximize the likelihood of the ground-truth target action, defined as the first action in the shortest path from the agent's current location $\boldsymbol{s}_t$ to the goal $\boldsymbol{s}_T^*$. In this regime, trajectories are generated by sampling an action from the policy with 50% probability, or selecting the ground-truth target action otherwise. In both sets of experiments we train all parameters end-to-end (except for the pretrained CNN). We have verified that the stand-alone performance of the filter is not unduly impacted by the addition of the policy, but we leave the investigation of more sophisticated RL training regimes to future work.

Implementation details. We provide further implementation details in section 1.6. PyTorch code is released to replicate all experiments.[2]

1.5 Experiments

1.5.1 Environment and dataset

Simulator. We use the Matterport3D Simulator [1] based on the Matterport3D dataset [17] containing RGB-D images, textured 3D meshes and other annotations captured from 11K panoramic viewpoints densely sampled throughout 90 buildings. Using this dataset, the simulator implements a visually-realistic first-person environment that allows the agent to look in any direction while moving between panoramic viewpoints along edges in a navigation graph. Viewpoints are 2.25m apart on average.

Depth outputs. As the Matterport3D Simulator supports RGB output only, we extend it

[2]https://github.com/batra-mlp-lab/vln-chasing-ghosts

to support depth outputs which are necessary to accurately project CNN features into the semantic spatial map. Our simulator extension projects the undistorted depth images from the Matterport3D dataset onto cubes aligned with the provided 'skybox' images, such that each cube-mapped pixel represents the euclidean distance from the camera center. We then adapt the existing rendering pipeline to render depth images from these cube-maps, converting depth values from euclidean distance back to distance from the camera plane in the process. To fill missing depth values corresponding to shiny, bright, transparent, and distant surfaces, we apply a simple cross-bilateral filter based on the NYUv2 implementation [38]. We additionally implement various other performance improvements, such as caching, which boosts the frame-rate of the simulator up to 1000 FPS, subject to GPU performance and CPU-GPU memory bandwith. We have incorporated these extensions into the original simulator codebase.[3]

R2R instruction dataset. We evaluate using the Room-to-Room (R2R) dataset for Vision-and-Language Navigation (VLN) [1]. The dataset consists of 22K open-vocabulary, crowd-sourced navigation instructions with an average length of 29 words. Each instruction corresponds to a 5–24m trajectory in the Matterport3D dataset, traversing 5–7 viewpoint transitions. Instructions are divided into splits for training, validation and testing. The validation set is further split into two components: val-seen, where instructions and trajectories are situated in environments seen during training, and val-unseen containing instructions situated in environments that are not seen during training. All the test set instructions and trajectories are from environments that are unseen in training and validation.

1.5.2 Goal prediction results

We first evaluate the goal prediction performance of our proposed mapper and filter architecture in a policy-free setting using fixed trajectories. Trajectories are generated by an agent that moves towards the goal with 50% probability, or randomly otherwise. As an

[3]https://github.com/peteanderson80/Matterport3DSimulator

Table 1.1: Goal prediction results given a natural language navigation instruction and a fixed trajectory that either moves towards the goal, or randomly, with 50:50 probability. We evaluate predictions at each time step, although on average the goal is not seen until later time steps. Our filtering approach that explicitly models trajectories outperforms LingUNet [11, 2] across all time steps (i.e., regardless of map sparsity). We confirm that add heading θ to the filter state provides a robust boost.

	Val-Seen									**Val-Unseen**								
Time step	0	1	2	3	4	5	6	7	Avg	0	1	2	3	4	5	6	7	Avg
Map Seen (m^2)	47.2	62.5	73.3	82.1	90.7	98.3	105	112	83.9	45.6	60.3	69.8	78.0	84.9	91.1	96.7	102	78.6
Goal Seen (%)	8.82	17.2	25.9	33.7	41.2	48.8	54.5	60.2	36.3	16.0	25.2	34.6	43.2	50.5	57.0	62.8	67.6	44.6
Prediction Error (m)																		
Hand-coded baseline	7.42	7.33	7.19	7.18	7.15	7.13	7.09	7.11	7.20	6.75	6.53	6.40	6.37	6.29	6.20	6.15	6.12	6.35
LingUNet baseline	7.17	6.66	6.17	5.75	5.42	5.15	4.89	4.69	5.74	6.18	5.80	5.40	5.17	4.90	4.65	4.44	4.27	5.10
Filter, $\boldsymbol{s} = (x, y)$ (ours)	6.45	5.94	5.66	5.25	5.00	4.86	4.67	4.62	5.31	5.92	5.50	5.14	4.88	4.67	4.45	4.41	4.30	4.91
Filter, $\boldsymbol{s} = (x, y, \theta)$ (ours)	**6.10**	**5.75**	**5.30**	**5.06**	**4.81**	**4.71**	**4.59**	**4.46**	**5.09**	**5.69**	**5.28**	**4.90**	**4.60**	**4.40**	**4.26**	**4.14**	**4.05**	**4.67**
Success Rate (<3m error)																		
Hand-coded baseline	17.3	17.8	18.5	18.2	18.0	19.1	18.8	18.6	18.3	18.9	20.1	21.1	21.3	21.8	22.2	22.6	22.9	21.4
LingUNet baseline	10.7	16.7	21.2	25.8	29.7	33.6	36.9	39.1	26.7	16.9	22.3	27.7	31.6	35.2	38.4	41.1	44.5	32.2
Filter, $\boldsymbol{s} = (x, y)$ (ours)	24.6	29.3	31.9	35.9	39.7	41.0	42.1	41.2	35.7	29.1	32.5	36.1	39.2	41.9	44.5	45.7	46.2	39.4
Filter, $\boldsymbol{s} = (x, y, \theta)$ (ours)	**30.9**	**34.3**	**38.4**	**41.6**	**43.7**	**44.9**	**44.3**	**46.2**	**40.6**	**34.2**	**38.7**	**42.7**	**46.1**	**48.2**	**48.4**	**49.9**	**51.2**	**44.9**

ablation, we also report results for our model excluding heading from the agent's filter state, i.e., $\boldsymbol{s}_t = (x, y)$, to quantify the value of encoding the agent's orientation in the motion and observation models. We compare to two baselines as follows:

LingUNet baseline. As a strong neural net baseline, we compare to LingUNet [2] – a language-conditioned variant of the U-Net image-to-image architecture [37] – that has recently been applied to goal location prediction in the context of a simulated quadrocopter instruction-following task [11]. We choose LingUNet because existing VLN models [3, 4, 5, 6, 7, 8, 9] do not explicitly model the goal location or the map, and are thus not capable of predicting the goal location from a provided trajectory. Following [11] we train a 5-layer LingUNet module conditioned on the sentence encoding $\boldsymbol{e}$ and the semantic map $\mathcal{M}$ to directly predict the goal location distribution (as well as a path visitation distribution, as an auxilliary loss) in a single forward pass. As we implement our observation model using a (smaller, 3-layer) LingUNet, the LingUNet baseline resembles an ablated single-step version of our model that dispenses with the decoder generating latent observations and actions as well as the motion model. Note that we use the same mapper architecture for our filter and for LingUNet.

Hand-coded baseline. We additionally compare to hand-coded goal prediction baseline designed to exploit biases in the R2R dataset [1] and the provided trajectories. We first calculate the mean straight-line distance from the start position to the goal across the entire training set, which is 7.6m. We then select as the predicted goal the position (x, y) in the map at a radius of 7.6m from the start position that has the greatest observed map area in an Gaussian-weighted neighborhood of (x, y).

Results. As illustrated in Table 1.1, our proposed filter architecture that explicitly models belief over trajectories that could be taken by a human demonstrator outperforms a strong LingUNet baseline at predicting the goal location (with an average success rate of 45% vs. 32% in unseen environments). This finding holds at all time steps (i.e., regardless of the sparsity of the map). We also demonstrate that removing the heading θ from the agent's state in our model degrades this success rate to 39%, demonstrating the importance of relative orientation to instruction understanding. For instance, it is unlikely for an agent following the true path to turn 180 degrees midway through (unless this is commanded by the instruction). Similarly, without knowing heading, the model can represent instructions such as 'go past the table' but not 'go past with the table on your left'. Finally, the poor performance of the handcoded baseline confirms that the goal location cannot be trivially predicted from the trajectory.

1.5.3 Vision-and-Language Navigation results

Having established the efficacy of our approach for goal prediction from a partial map, we turn to the full VLN task that requires our agent to take actions to actually reach the goal.

Evaluation. In VLN, an episode is successful if the final navigation error is less than 3m. We report our agent's average success rate at reaching the goal (`SR`), and `SPL` [16], a recently proposed summary measure of an agent's navigation performance that balances navigation success against trajectory efficiency (higher is better). We also report trajectory length (TL) and navigation error (NE) in meters, as well as oracle success (OS), defined as the agent's success rate under an oracle stopping rule.

Results. In Table 1.2, we present our results in the context of state-of-the-art methods; however, as noted by the RL and Aug columns in the table, these approaches include reinforcement learning and complex data augmentation and pretraining strategies. These are non-trivial extensions that are the result of a community effort [3, 4, 5, 6, 7, 8, 9] and are orthogonal to our own contribution. We also use a less powerful CNN (ResNet-34 vs. ResNet-152 in prior work). For the most direct comparison, we consider the ablated models in the lower panel of Table 1.2 to be most appropriate. We find these results promising given this is the first work to explore such a drastically different model class (i.e., maintaining a metric map and a probability distribution over alternative trajectories in the map). Our model also exhibits less overfitting than other approaches – performing equally well on both seen (val-seen) and unseen (val-unseen) environments.

Further, our filtering approach allows us greater insight into the model. We examine a qualitative example in Figure 1.3. On the left, we can see the agent attends to appropriate visual and direction words when generating latent observations and actions, supporting the intuition in Figure 1.1. On the right, we can see the growing confidence our goal predictor places on the correct location as more of the map is explored – despite the increasing number of visible alternatives.

1.6 Implementation Details

Simulator. In experiments, we set the Matterport3D simulator [1] to generate 320×256 pixel images with a 60 degree vertical field of view. To capture more of the floor and nearby obstacles (and less of the roof) we set the camera elevation to 30 degrees down from horizontal. At each panoramic viewpoint location in the simulator we capture a horizontal sweep containing 12 images at 30 degree increments, which are projected into the map in a single time step as described in subsection 1.4.1.

Table 1.2: Results for the full VLN task on the R2R dataset. Our model achieves credible results for a new model class trained exclusively with imitation learning (no `RL`) and without any data augmentation or specialized pretraining (`Aug`).

			Val-Seen					Val-Unseen					Test				
Model	RL	Aug	TL (m)	NE (m)	OS (%)	SR (%)	SPL	TL (m)	NE (m)	OS (%)	SR (%)	SPL	TL (m)	NE (m)	OS (%)	SR (%)	SPL
RPA [4]	✓		8.46	5.56	53	43	-	7.22	7.65	32	25	-	9.15	7.53	32	25	23
Speaker-Follower [7]		✓	-	3.36	74	66	-	-	6.62	45	36	-	14.82	6.62	44	35	28
RCM [3]	✓		10.65	3.53	75	67	-	11.46	6.09	50	43	-	11.97	6.12	50	43	38
Self-Monitoring [5]		✓	-	3.18	77	68	58	-	5.41	59	47	34	18.04	5.67	59	48	35
Regretful Agent [6]		✓	-	3.23	77	69	63	-	5.32	59	50	41	13.69	5.69	56	48	40
FAST [9]	✓		-	-	-	-	-	21.1	4.97	-	56	43	22.08	**5.14**	**64**	**54**	41
Back Translation [8]	✓	✓	11.0	3.99	-	62	59	10.7	5.22	-	52	48	11.66	5.23	59	51	**47**
Speaker-Follower [7]			-	4.86	63	52	-	-	7.07	41	31	-	-	-	-	-	-
Back Translation [8]			10.3	5.39	-	48	46	9.15	6.25	-	44	40	-	-	-	-	-
Ours			10.15	7.59	42	34	30	9.64	7.20	44	35	31	10.03	7.83	42	33	30

Mapper. For our CNN implementation we use a ResNet-34 [39] architecture that is pre-trained on ImageNet [40]. We found that fine-tuning the CNN while training our model mainly improved performance on the Val-Seen set, and so we left the CNN parameters fixed in the reported experiments. To extract the visual feature representation $\boldsymbol{v}$ we concatenate the output from the CNN's last 2 layers to provide a $16 \times 20 \times 768$ representation. The dimensionality of our map representation $\mathcal{M}$ is fixed at $128 \times 96 \times 96$ and each cell represents a square region with side length 0.5m (the entire map is thus 48m $\times$ 48m). In the mapper's convolutional [31] GRU [32] we use 3×3 convolutional filters and we train with spatial dropout [41] of 0.5 in both the input-to-state and state-to-state transitions with fixed dropout masks for the duration of each episode.

Filter. In the instruction encoder we use a hidden state size of 256 for both the forward and backward encoders, and a word embedding size of 300. We use a motion kernel size M of 7, but we upscale the motion kernel $g(\boldsymbol{a}_t, \mathcal{M})$ by a scale factor of $2\times$ before applying it such that the agent can move a maximum of 3.5m in a single time step.

Training. In training, we use the Adam optimizer [42] with an initial learning rate of 1e-3, weight decay of 1e-7, and a batch size of 5. In the goal prediction experiment, all models are trained for 8K iterations, after which all models have converged. In the full VLN experiment, our models are trained for 17.5K iterations, and we pick the iteration with the highest SPL

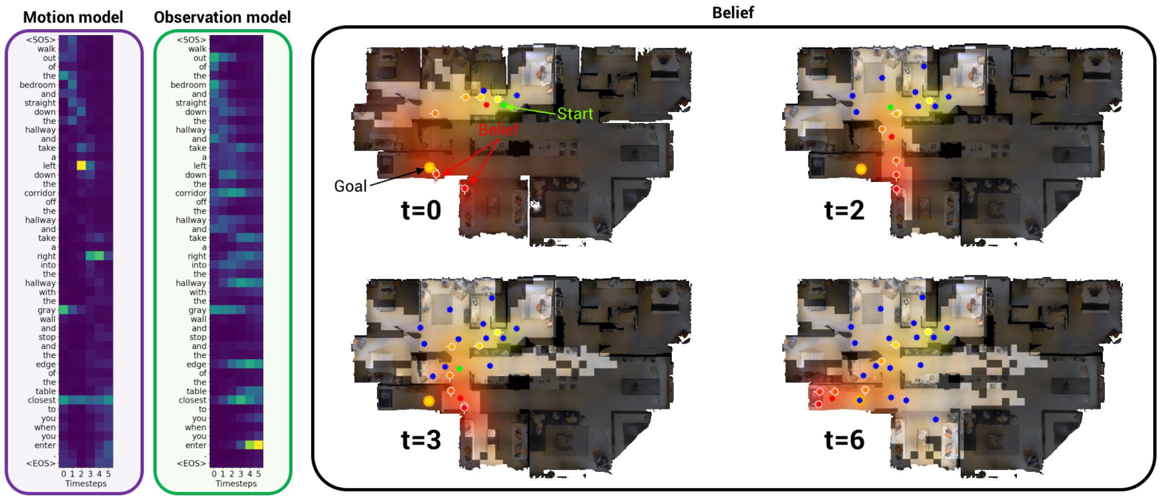


Figure 1.3: Left: Textual attention during latent observation and action generation is appropriately more focused towards action words (‘left’, ‘right’) for the motion model, and visual words (‘bedroom’, ‘corridor’, ‘table’) for the observation model. Right: Top-down view illustrating the agent’s expanding semantic spatial map (lighter-colored region), navigation graph (blue dots) and corresponding belief (red heatmap and circles with white heading markers) when following this instruction. At $t = 0$ the map is largely unexplored, and the belief is approximately correct but dispersed. By $t = 6$, the agent has become confident about the correct goal location, despite many now-visible alternative paths.

performance on Val-Unseen to report and submit to the test server. Training the model takes around 1 day for goal prediction, and 2.5 days for the full VLN task, using a single Titan X GPU.

Visualizations. In Figure 1.3, we depict top-down floorplan visualizations of Matterport environments to provide greater insight into the model’s behavior. These visualizations are rendered from textured meshes in the Matterport3D dataset [17], using the provided GAPS software which was modified to render using an orthographic projection.

1.7 Visualizations

Observations and actions. In this section we provide further visualizations of the attention weights in the sequence decoders that generate latent observations and actions (refer to subsection 1.4.2). Instructions are examples from the Val-Unseen set. In general, the attention models for the motion model (generating latent action vectors $\boldsymbol{a}$) and the observation model (generating latent observation vectors $\boldsymbol{o}$) specialize in different ways. The motion model focuses attention on action words, while the observation model focuses on visual words, as illustrated in Figure 1.4 and Figure 1.5. The sequential ordering of the instructions (e.g., attention weights showing a diagonal structure from top-left to bottom-right) is also evident.

1.8 Conclusion

We show that instruction following can be formulated as Bayesian state tracking in a model that maintains a semantic spatial map of the environment, and an explicit probability distribution over alternative possible trajectories in that map. To evaluate our approach we choose the complex problem of Vision-and-Language Navigation (VLN). This represents a significant departure from existing work in the area, and required augmenting the Matterport3D simulator with depth. Empirically, we show that our approach outperforms recent alternative approaches to goal location prediction, and achieves credible results on the full VLN task without using RL or data augmentation – while offering reduced overfitting to seen environments, unprecedented intepretability and less reliance on the simulator's navigation constraints.

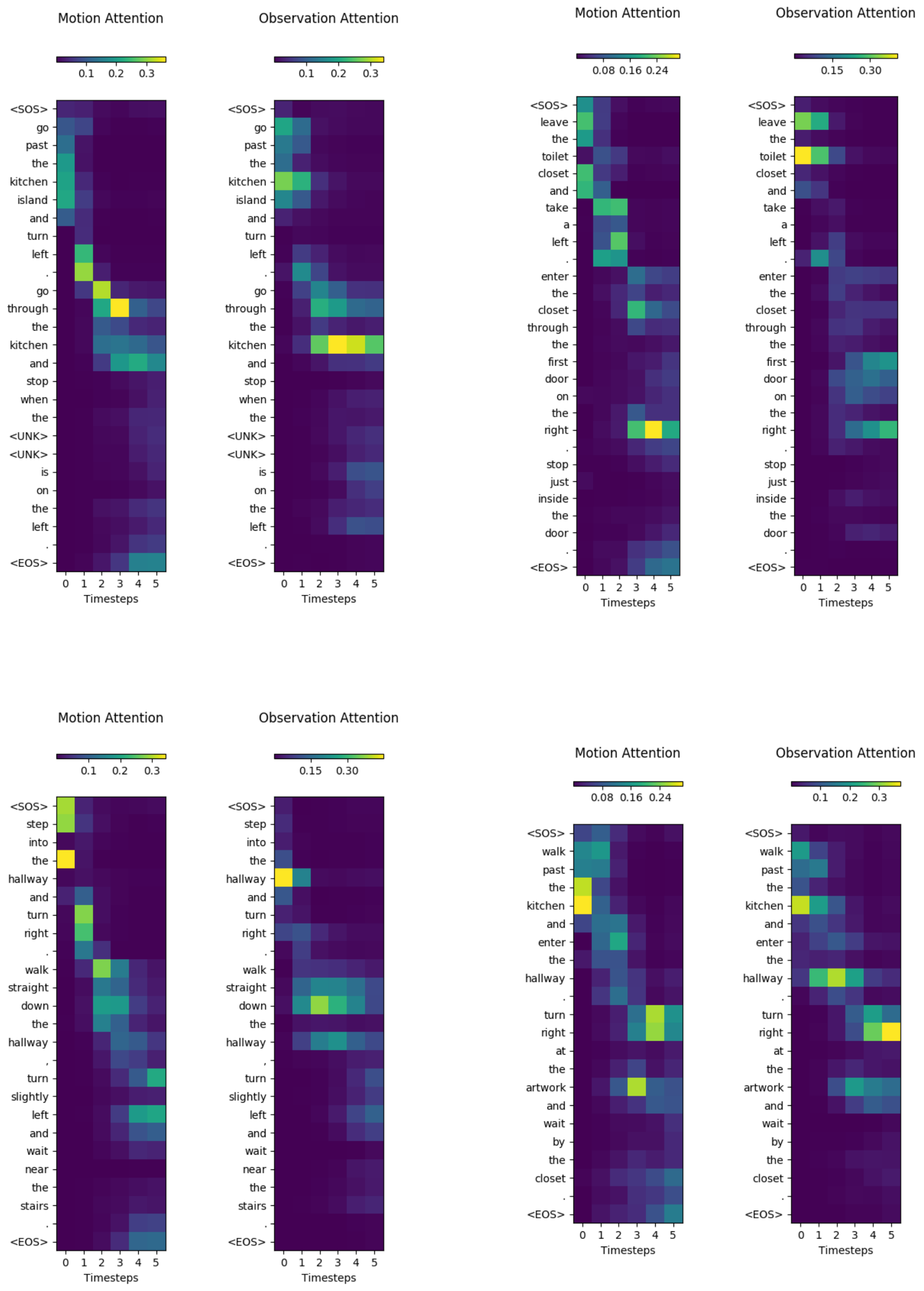


Figure 1.4: Example attention weight visualizations from the motion model and observation model inputs. The motion model focuses on action descriptions, such as ‘go through’ (top left) and ‘turn right’ (bottom left and right). In contrast, the observation model focuses more towards visual words such as ‘kitchen’ (top left), ‘toilet’ (top right), and ‘hallway’ (bottom left and right).

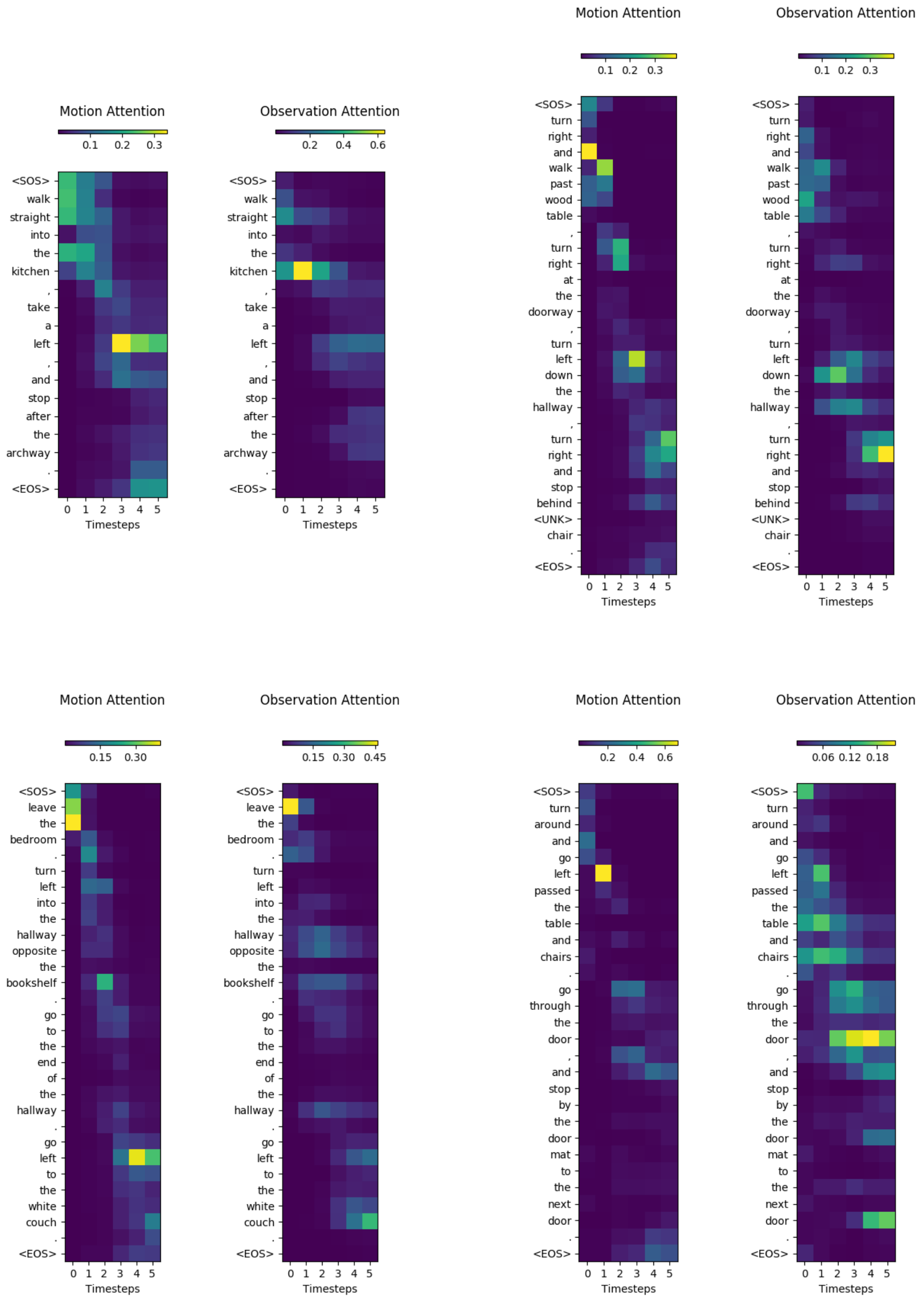


Figure 1.5: More example attention weight visualizations from the motion and observation models. Here, the motion model focuses on action descriptions, such as ‘left’, while the observation model focuses attention on visual words such as ‘couch’ (bottom left), ‘table and chairs’ (bottom right), and ‘door’ (bottom right).

CHAPTER 2
SIM-TO-REAL TRANSFER FOR VISION-AND-LANGUAGE NAVIGATION

2.1 Introduction

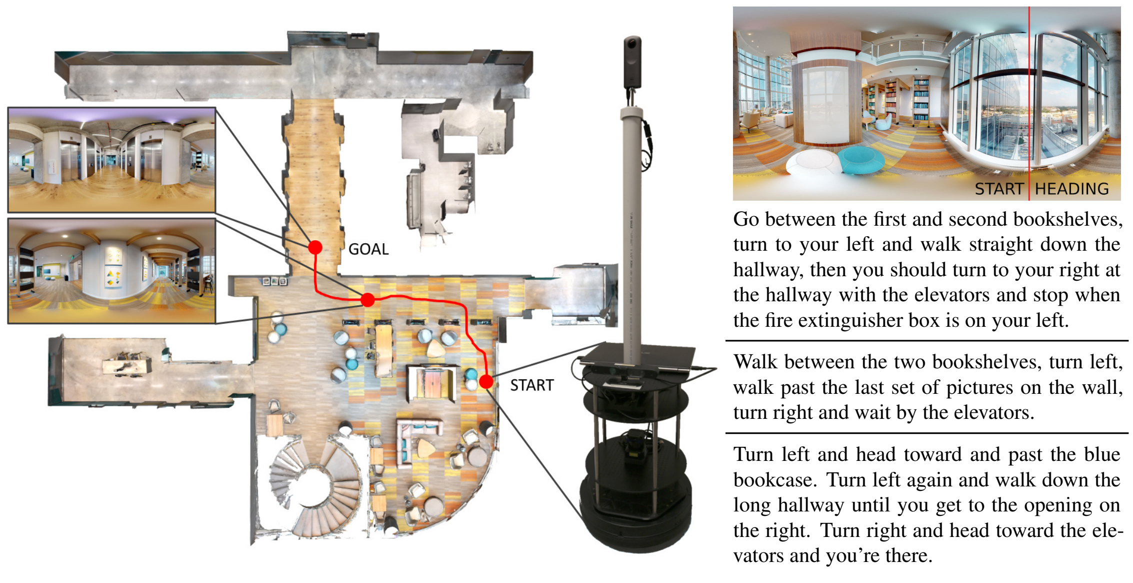
We study the challenging problem of releasing a robot in a previously unseen environment, and having it follow unconstrained natural language navigation instructions. Most previous evaluations of instruction-following robots either focus on smaller table-top environments [43, 44, 45], or are evaluated in simulation [46, 47, 48, 49, 50, 51, 52, 53, 54]. However, performing only component-level evaluation (e.g., of the instruction parser) or evaluating only in simulation neglects real-world sensing, actuation and localization errors and the challenges of integrating complex components, which may give a misleading impression of progress. Therefore, leveraging the cumulative advances of previous authors studying Vision-and-Language Navigation (VLN) in simulation [1, 3, 4, 5, 6, 7, 8, 9], we transfer a VLN agent trained in simulation on the R2R dataset [1] to a physical robot and complete one of the first full system evaluations of a robot following unconstrained English language directions in an unseen building.

As illustrated in Figure 2.1, VLN [1] is a formulation of the instruction-following problem that requires an agent to interpret a natural-language instruction and then execute a sequence of actions to navigate efficiently from the starting point to the goal in a previously unseen environment. In existing working studying VLN in simulation, the agent's action space is typically defined in terms of edge traversals in a navigation graph, where nodes are represented with 360° panoramic images (on average 2.1m apart) and edges indicate navigable paths between these panoramas. In effect, high visual fidelity comes at the cost of low control fidelity. Given this limitation, a major challenge with sim-to-real transfer of a VLN agent is bridging the gap between the high-level discrete action space learned by the agent and the low-level continuous physical world in which the robot operates.

To address this challenge, we propose a subgoal model that, conditioning on both 360° RGB images and laser scans, identifies a set of navigable nearby waypoints that can be evaluated by the VLN agent as high-level action candidates. To minimize domain differences between sim and real, the subgoal model is trained on the same navigation graphs (from the R2R / Matterport3D [17] dataset) as the VLN agent. To support navigation to the waypoint selected by the VLN agent, we assemble a classical navigation stack based on Robot Operating System (ROS) [55], incorporating a standard Simultaneous Localization and Mapping (SLAM) implementation [56] along with obstacle avoidance and path-planning routines. For sim-to-real experiments we use a TurtleBot2 mobile robot equipped with a 2D laser scanner and a 360° RGB camera. To mitigate the impact of visual differences between sim and real, we train the VLN agent and the subgoal model using domain randomization [57].

Evaluations are conducted in a 325m^2 physical office environment we refer to as Coda. Since our focus is evaluating whether progress on VLN in simulation can be parlayed into progress in robotics, we digitize and annotate the Coda environment to provide a reliable comparison of sim and real performance in parallel unseen test environments. Specifically, a Matterport camera is used to scan and reconstruct Coda and the resulting assets are imported into the Matterport3D simulator. We then construct a dataset of English language instruction-trajectory pairs in Coda using Amazon Mechanical Turk (AMT) and following the R2R data collection protocols. In total, we collect 111 navigation instructions representing 1,334m of language-guided trajectories.

Experimentally, we complete two full physical evaluations in Coda, testing two settings: ‘with map’ in which an occupancy map and navigation graph are collected and annotated in advance, and ‘no map’ in which the robot performs SLAM from scratch each time a new instruction is received. In all evaluations the language inputs and the visual appearance of the environment are previously unseen. We show that sim-to-real transfer is reasonably



Go between the first and second bookshelves, turn to your left and walk straight down the hallway, then you should turn to your right at the hallway with the elevators and stop when the fire extinguisher box is on your left.

Walk between the two bookshelves, turn left, walk past the last set of pictures on the wall, turn right and wait by the elevators.

Turn left and head toward and past the blue bookcase. Turn left again and walk down the long hallway until you get to the opening on the right. Turn right and head toward the elevators and you're there.

Figure 2.1: We transfer a VLN agent trained in simulation to a physical robot (center) placed in a 325m^2 office environment not seen in training (left). Our experiments compare instruction-following performance in parallel simulator and physical environments over 111 unconstrained natural language navigation instructions (e.g., the 3 instructions on right corresponding to the red target trajectory).

successful in the 'with map' setting (success rate of 46.8% vs. 55.9% in simulation, with reductions in success rate attributed 3.9% to remaining visual domain differences and 5.2% to viewpoint differences). However, in the hardest setting with no prior mapping, sim-to-real transfer is much less reliable (success rate of 22.5%) due to subgoal prediction errors which fail to fully abstract the differences in the agent's action space between sim and real.

Contributions. In summary, we achieve the first sim-to-real transfer of a VLN agent trained in simulation on the R2R dataset to a robotic platform. Our main contributions include:

- A new annotated VLN simulator environment corresponding to an accessible physical environment,
- A sim-to-real framework interfacing a trained VLN agent with standard ROS components,
- A subgoal prediction model to bridge the gap between the discrete action space learned by the VLN agent and the continuous world, and

- An empirical study quantifying the sim-to-real gap in terms of errors due to visual domain differences, viewpoint differences and subgoal prediction errors / action space differences.

We provide code[1] for digitizing and annotating new VLN environments, plus our sim-to-real framework and subgoal model for deploying VLN agents to robots using ROS.

2.2 Related Work

Instruction-Following Robots. Although natural language command of robots in unstructured environments could be considered a grand challenge of robotics, there is surprisingly little previous work in our setting of interest – in which a physical robot is released in a building it hasn't seen in training and evaluated on it's ability to execute unconstrained natural language navigation instructions. The most similar settings to ours are (noting major differences from ours in parentheses): the voice-commandable wheelchair of [58] (relies on artificial landmarks), the quadcopters of [59] and [60] (evaluated in the training environment), and the voice-commanded robot teammate of [61] (outdoors). Most other evaluations of language-guided robots have been limited to only a handful of instruction commands [62, 63] or a handful of object types [64, 65], or they focus on manipulation rather than navigation [66, 67].

Vision-and-Language Navigation (VLN). In the VLN task [1], an agent is placed in a photo-realistic simulation of an indoor environment and given a natural language navigation instruction describing the path to the goal. To reach it, agents must learn to ground language instructions to both visual observations and actions. In the standard setting, the test environments are unseen during training and no prior exploration is permitted. Despite the task's difficulty, recent work has seen significant improvements [3, 4, 5, 6, 7, 8, 9], including the use of pragmatic speaker models for trajectory re-ranking and data augmentation [7, 8], as well as progress estimation [5] and backtracking [6, 9].

[1]https://github.com/batra-mlp-lab/vln-sim2real

Pano-Simulators and Datasets. A growing number of simulation environments, tasks and datasets have been proposed based on situated panoramic images. For example, building on R2R/Matterport3D [1, 17], annotations for vision-and-dialog navigation [68], asking for help [69], remote embodied referring expressions [70], and multilingual VLN [71, 72] have been released. In the outdoor setting, several panoramic image datasets have been proposed including StreetLearn [73, 74] and SEVN [75], giving rise to language navigation datasets such as TouchDown [76], Talk2Nav [77] and RUN [78]. With the increasing interest in training embodied agents in panoramic image environments, there is an urgent need to investigate the transfer of these agents to real physical platforms.

2.3 Sim-to-Real Experimental Setting

2.3.1 Coda Test Environment

For evaluation of the VLN robot we select Coda as the unseen test environment. Coda is a 325m^2 collaborative shared space in a commercial office building. As a shared space, Coda is devoid of personal items such as papers, posters, photos, bags and computing devices. While this eliminates some interesting visual clutter, it also helps minimize the drift between the static simulator and the real physical environment over time. Visual diversity is enhanced by the variety of rooms included, such as an elevator lobby, several long corridors, a lounge area and various break-out spaces (refer floorplan in Figure 2.1), while floor-to-ceiling glass walls and windows provide reflections and changing lighting that makes the space particularly challenging for robotic vision.

2.3.2 Simulator Construction

To accurately establish the sim-to-real gap in the unseen test environment, we construct a parallel simulator environment by reconstructing Coda with a Matterport3D Pro 2 camera and the Matterport3D web services. We download the resulting pointcloud and textured mesh, plus the equirectangular panoramic image and pose at each of the camera 65 viewpoints, and a ‘visibility graph’ indicating which pairs of camera viewpoints are mutually visible. After excluding 6 viewpoints located on stairs or above furniture (which are unreachable by our robot), following [1] we construct a navigation graph from the visibility graph by excluding edges with length greater than 5m. The resulting navigation graph, panoramic images and viewpoint poses are then imported into the Matterport3D simulator to create the Coda simulator environment.

2.3.3 Navigation Instructions

To collect navigation instructions for Coda, we follow [1] by sampling trajectories that are the shortest path between two points and then asking annotators to describe these paths using an immersive 3D web interface. To validate instruction quality, we then ask a different annotator to follow each instruction using a similar interface. In total we sample 37 trajectories and collect four English language navigation instructions for each trajectory using Amazon Mechanical Turk (AMT). After discarding the instruction with the highest follower navigation error for each trajectory, the final Coda dataset contains 111 instructions, representing a total of 1,334m of language-guided navigation with an average human follower success rate of 93% (vs. 86% for the R2R test set). As indicated by the example in Figure 2.1, the resulting trajectories and instructions are similar in length and style to R2R, with an average of 25 words per instruction (vs. 29 for R2R). The vocabulary size of the Coda instructions is in the 44th percentile of R2R environment vocabularies (based on repeatedly randomly sampling 37 paths / 111 instructions per R2R environment, and skipping R2R environments with insufficient instructions). This suggests that the language is as diverse as many of the environments in the original dataset. Qualitatively, the instructions mention a variety of objects (e.g., water fountain, art, sectional sofa), along with attributes such as color (12 mentioned), state (e.g., open, closed, hanging), size (e.g., small, big), and composition (e.g., wood, glass, cement) using both allocentric and egocentric reference frames.

2.3.4 Robot Platform

We conduct experiments using a low-cost TurtleBot2 robot consisting of a Kobuki mobile base, an Asus Xion Pro Live RGBD camera, and an Asus netbook. Since virtually all VLN agents have been developed using 360° vision, we additionally equip the robot with a Ricoh Theta V 360° consumer-grade RGB camera. For the most direct sim-to-real comparison we mount the Theta V camera 1.35m above ground level – the same height we set the Matterport camera tripod when scanning and reconstructing Coda. Lastly, for obstacle avoidance and mapping we mount a 270° Hokuyo 2D laser scanner with 30m range at 0.24m above ground level. The robot runs ROS-kinetic [55]. We use standard ROS / TurtleBot packages such as gmapping, amcl and move_base as well as PyTorch [79]. During evaluation all code executes on the robot, except PyTorch ROS nodes (the VLN agent and our subgoal model) which are run remotely.

2.3.5 Evaluation Metrics.

We use standard VLN metrics for evaluation in both sim and real, with the aim of testing whether performance on the robot can match the results in simulation for the same unseen test environment. In all experiments the agent must terminate the episode as near to the goal position as possible. An episode is considered successful if the navigation error (defined as the distance between the agent's final position and the goal position) is less than 3m. We report average values for trajectory length (`TL`), navigation error (`NE`), success rate at reaching the goal (`SR`), oracle success rate (`OS`) – defined as the agent's success rate at the closest point on the agent's trajectory to the goal, Success weighted by (normalized inverse) Path Length (`SPL`) [16] and both Normalized and Success-weighted Dynamic Time Warping (`NDTW` and `SDTW`) [80]. `SPL` and `SDTW` are summary measures of navigation performance that balance success against trajectory efficiency and fidelity (i.e., similarity to the ground-truth path), while `NDTW` measures path fidelity irrespective of success (higher is better for each). When calculating `NDTW` and `SDTW` we represent both sim and robot trajectories using 100 equally spaced points to eliminate sampling differences.

2.3.6 Robot Pose Tracking

To compute these evaluation metrics we must know the agent's pose at all points in time. In the simulation this is available. To track the robot's pose, we first teleop the robot through Coda to construct a 0.05m resolution occupancy map using the laser scanner and the ROS gmapping SLAM package. We then register this map to the Matterport3D coordinate frame. For evaluation purposes we use the particle filter provided by the ROS amcl package to track the robot's pose during experiments. We estimate that the error in this pose estimate is typically an order of magnitude less than the 3m radius used for determining success.

2.4 Adapting a VLN Agent to a Physical Robot

2.4.1 VLN Agent

We now detail our approach to sim-to-real transfer of a VLN agent onto the robot. We start with an existing VLN agent [8] that achieved state-of-the-art performance on the unseen environments in the R2R test split. Typical of most recent work on this task, this agent is trained using a (simulated) panoramic 360° RGB sensor, processing the entire visual context at each step, and a highly-abstracted discrete action space provided by the Matterport3D simulator. Actions are defined by neighboring panoramic image viewpoints, which are on average 2.1m away. The agent processes the visual representations in the direction of those locations, and then once an action has been selected the simulator teleports the agent to the new viewpoint (refer Figure 2.2). To transfer this agent to a robot we must address differences in the visual domain and action space, plus navigation and localization errors that are not experienced in training. We discuss these challenges in turn.

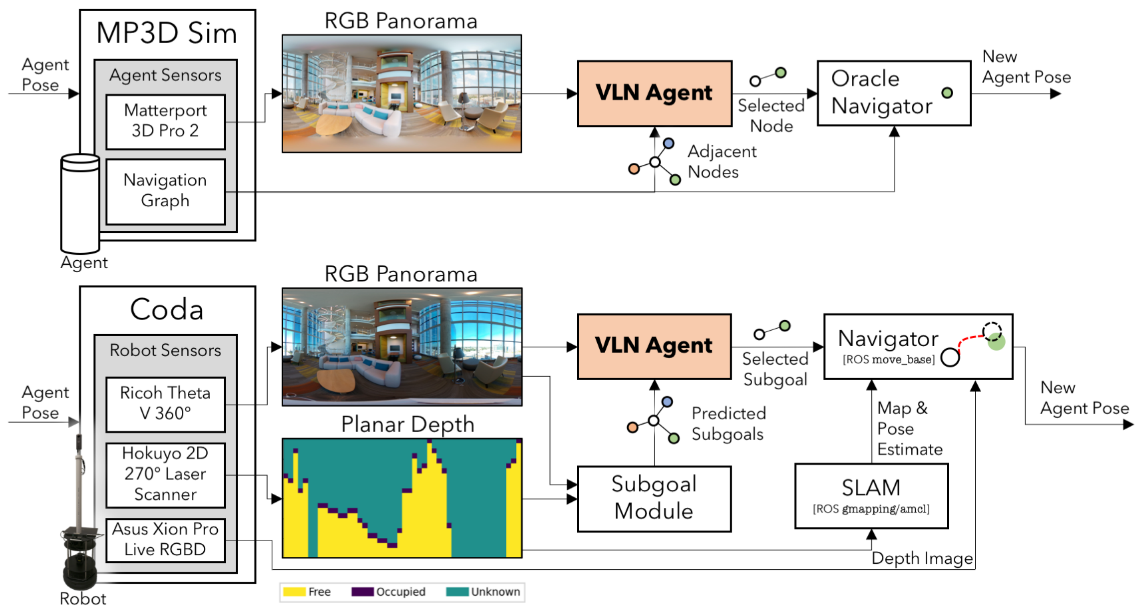


Figure 2.2: We evaluate a VLN agent trained in simulation in matching previously unseen sim (top) and real (bottom) environments. Our sim-to-real framework includes domain randomization, a subgoal model to bridge the action space differences between sim and real (removing the reliance on the simulator navigation graph), and standard ROS components for SLAM and navigation.

2.4.2 Visual Domain Adaptation

As illustrated by the RGB panorama differences in Figure 2.2, the robot's Ricoh Theta V camera is consumer-grade with limited dynamic range compared to the Matterport camera, resulting in a loss of visual detail and a 6.4% reduction in success rate due to visual domain differences (refer section 2.5). To address this problem, domain adaptation algorithms typically require a large set of target domain images – in this case Theta V panoramic images captured in a variety of indoor environments – which are non-trivial to collect. In preliminary experiments, simple alternatives such as histogram matching to align the Theta V colorspace with the Matterport camera were not effective. We therefore selected an approach akin to domain randomization [57], in which we applied random color jitter to each panorama when training the VLN agent in the Matterport simulator. Brightness, contrast and saturation were varied by a factor of 0.3 and hue was varied by a factor of 0.01, with these parameters determined through visual inspection.

2.4.3 Action Space Adaptation

To accommodate the transfer from the highly-abstracted action space of the simulator, which is based on a navigation graph, to the primitive motor control actions of a physical robot, we propose to use a *subgoal model* in conjunction with standard ROS navigation modules. After conditioning on available sensor observations, the subgoal model predicts a set of nearby waypoints, or subgoals, for the VLN agent to choose from. In effect, we provide the VLN agent with an implementation of the simulator's action space and depend on ROS to execute those actions. Our approach is therefore a modular fusion of both classical and learning-based methods.

As illustrated in the example in Figure 2.3 left, as input to the subgoal model we use the most recent laser scan, to provide an approximate indication of free space, and visual inputs to fill in the gaps and add additional context. We represent the laser scan as a radial occupancy map over range and heading bins, and the visual input using pretrained ResNet-152 [81] CNN features captured at 12 different headings and 3 different camera evaluations (the same visual feature representation consumed by the VLN agent). The subgoal model is based on a 4-stage U-Net [37] architecture that takes the radial occupancy map as input and fuses the visual features in the 3rd downsampling stage. The output of the model is the probability that each laser scan heading and range bin contains a waypoint.

The subgoal model is trained and validated on viewpoints from the Matterport3D dataset [17]. Since the dataset does not include planar laser scans, we simulate the output of a 270° laser scan at each viewpoint by measuring the range to the reconstructed matterport mesh. The model is trained to minimize the sinkhorn divergence [82] between predicted waypoint locations and the location of neighboring viewpoints in the navigation graph, after removing neighboring viewpoints that would require traversing stairs. Sinkhorn divergence is a smoothed approximation to earth mover's distance that we find to be more effective

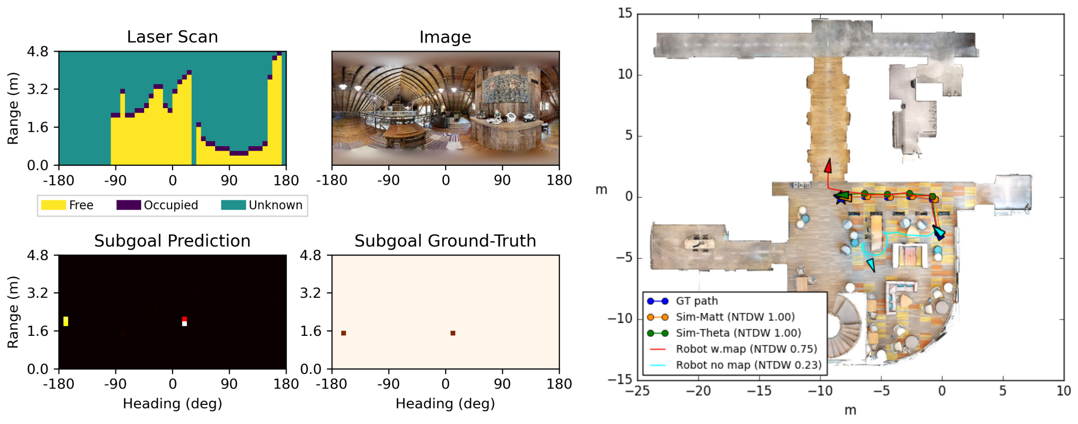


Figure 2.3: Left: To predict the next set of potential waypoints, or subgoals, we combine a radial occupancy map representation of the laser scan (top-left) with pretrained CNN features from the panoramic image (top-middle) in a U-Net [37] architecture. Even with missing range data (green regions) due to the 270° scan, the model generally predicts plausible subgoals learned from the Matterport simulator navigation graphs (bottom-left vs. bottom-middle). However, when the subgoal model fails to predict a valid waypoint (right, where the subgoal model does not predict a waypoint passing between narrow bookshelves) then the robot navigation fails.

than cross-entropy, which does not respect the underlying metric space. At test time in the Coda environment we use a confidence threshold to select the final set of waypoints (up to a maximum of 5) that are provided to the VLN agent. Using the Theta V camera and evaluating at each viewpoint in the Coda navigation graph we find that 29% of predicted waypoints are within 0.5m of a ground-truth neighboring viewpoint (60% within 1m, 74% within 1.5m).

2.4.4 Obstacle Avoidance and Path Planning

For obstacle detection we rely on the depth camera and the 270° laser scanner. These sensors are complimentary since, although the camera only has a narrow horizontal field of view, it senses overhanging obstacles such as table tops that may be missed in the narrow planar sweep of the laser scanner (and could be impacted by our tall robot). We rely on the ROS move_base navigation module to integrate these observations, maintain local and global costs maps, and to plan and execute motion to the waypoint selected by the VLN model.

2.4.5 Verification

Porting an existing VLN agent to ROS required significant refactoring of the original codebase. To verify our implementation we created mock ROS nodes for the robot's panoramic camera, the subgoal model, and the move_base navigation package using images from the simulator and the simulator navigation graph. This allowed us to run our robot code on simulator data and verified that the performance matched the original R2R-EnvDrop model [8] within 1%. Given the large number of different VLN agents and architectures that have been proposed [3, 4, 5, 6, 7, 8, 9], we hope that our framework will ease the sim-to-real burden in future work. As shown in Figure 2.2, this codebase acts as a ROS-based harness around the standard VLN agent api adopted by the community.

2.5 Results and Analysis

In this section, we report sim-to-real results and analysis for the VLN robot. We first characterize the difficulty of the Coda environment relative to R2R, and examine the importance of visual domain adaptation. We then consider two sim-to-real settings: the first in which the robot is provided with a laser-scanned occupancy map of the environment and the simulator navigation graph ('with map'), and the second in which the environment is completely unseen at the beginning of each episode, i.e., there is no provided map or navigation graph and the robot's SLAM map is reset each time an instruction is received ('no map'). No aspect of our system is trained or validated in the Coda environment (in simulation or in reality) in either setting. All experiments are conducted in daylight and furniture is restored to its original position before each evaluation. This does not assist the robot (which, in our hardest setting, experiences the environment as previously unseen each time an instruction is received). Rather, this minimizes drift between the simulator and the physical environment, enabling us to more accurately characterize sim-to-real performance.

2.5.1 How challenging is Coda?

We first establish the relative difficulty of the Coda environment compared to existing environments in the Matterport3D [17] / R2R [1] dataset. We report the performance of the R2R-EnvDrop [8] VLN agent in the Coda simulator compared to R2R val and test. As illustrated in Table 2.1, after training on the R2R training set we achieved a 49.9% success rate on the R2R val-unseen set, and a 49.2% success rate on R2R test. On Coda (sim), the agent's success rate is slightly higher at 55.9%. This suggests that, although the Coda dataset was collected by a different camera operator, and a different pool of AMT workers at a different time to R2R, the collection protocol has been faithfully followed and the dataset is 'in-domain' with respect to R2R. Accordingly, we treat Coda sim performance (row 4) as a baseline to investigate sim-to-real transfer.

	Split	`TL` (m)	`NE` (m)	`OS` (%)	`SR` (%)	`SPL`	`SDTW`	`NDTW`
1	R2R Val-Seen	9.70	4.56	62.0	57.5	55.0	49.6	60.8
2	R2R Val-Unseen	9.18	5.42	55.4	49.9	47.0	44.5	55.2
3	R2R Test	9.52	5.57	54.9	49.2	46.8	-	-
4	Coda	11.22	4.98	59.5	55.9	53.6	44.0	54.9

Table 2.1: Characterizing the difficulty of the Coda simulator environment using the EnvDrop agent [8]. Performance in Coda is slightly higher than the R2R test set on average. R2R Val-Unseen, R2R Test and Coda are previously unseen visual environments.

2.5.2 How important is visual domain adaptation?

To address this question, we replaced the Matterport panoramic images in the Coda simulator using images captured with the robot's Ricoh Theta V camera from the same viewpoint locations. This allows us to isolate the importance of visual domain adaptation in the absence of other factors. As foreshadowed in Figure 2.2, switching to the cheaper camera (row 5 vs. row 4 in Table 2.2) causes a 6.4% drop in success rate and a 6.2% drop in SPL, which is reduced to a 3.9% and 4.3% drop respectively (row 6 vs. row 4) when we retrained the agent using visual domain adaptation. To make these results as reliable as possible, rows 5 and 6 are averaged over 3 sets of Theta V images that were captured on different days, exhibiting a variation in success rate of ±1.9% without domain adaptation and ±1.4% with domain adaptation.

	Setting	Camera	Adapted	Map	**TL** (m)	**NE** (m)	**OS** (%)	**SR** (%)	**SPL**	**SDTW**	**NDTW**
4	Sim	Matterport	-	-	11.22	4.98	59.5	55.9	53.6	44.0	54.9
5	Sim	Theta V	✗	-	11.21	5.71	55.3	49.5	47.4	39.5	52.0
6	Sim	Theta V	✓	-	11.38	5.74	59.2	52.0	49.3	42.6	54.0
7	Robot	Theta V	✓	✓	11.32	6.04	51.4	46.8	43.9	28.7	38.5
8	Robot	Theta V	✓	✗	8.02	6.56	26.1	22.5	21.9	13.8	30.0

Table 2.2: Sim-to-real performance comparison for our VLN agent in Coda over 1,334m of language-guided trajectories. Success rate at reaching the goal (**SR**) and path fidelity (**NDTW**) remains relatively high in the robot 'with map' setting (row 7), but is reduced in the 'no map' setting (row 8).

2.5.3 How large is the sim-to-real gap with a map?

In the 'with map' setting, we conduct a full evaluation on the robot while also providing a pre-captured laser scan to assist with obstacle avoidance and path-planning, and the simulator navigation graph to provide waypoint candidates. The subgoal model is not used. We consider this setting for two reasons. First, for a robot operating in a single environment it may be reasonable to provide some navigation annotations and to expect the robot to maintain an occupancy map. Second, this setting tests the implicit assumption in graph-based simulators that existing robotics systems are capable of navigating between viewpoints in the graph.

As reported in Table 2.2, in the 'with map' setting the robot achieves an instruction-following success rate of 46.8% in physical Coda, a reduction of 5.2% compared to using the robot camera in simulation (row 7 vs row 6). During this evaluation over 111 instructions and 1.3km of trajectories we did not experience any collisions or navigation failures, which we define as the robot failing to navigate an edge between panoramic viewpoints in the navigation graph. For context, the environment contains five ‘squeeze points’ with 80-90cm gaps between furniture, compared to which the width of the robot with laptop is around 40cm. We attribute the reduction in performance to a degradation of visual inputs from viewpoints that are slightly out-of-position compared to the simulator. We also note that, despite our best efforts, there were some people present in Coda at times during testing, which may also degrade robot performance due to domain differences (images of people are extremely rare in the Matterport3D dataset). Overall we conclude that, at least based on analysis performed in the Coda environment, if an occupancy map and navigation graph are collected and annotated in advance, transferring a VLN agent to an inexpensive robot using a classical navigation stack is feasible with around a 9.1% reduction in success rate (row 7 vs. row 4).

2.5.4 How large is the sim-to-real gap overall?

In the final experimental setting, we revoke the robot’s access to the Coda occupancy map and the simulator navigation graph. Removing the navigation graph requires the robot to rely on waypoint predictions from the subgoal model for the first time. Removing the occupancy map requires the robot to perform simultaneous localization and mapping (SLAM) from scratch in each episode (every time a new instruction is received), which makes obstacle avoidance and path planning more challenging.[2] Overall, this setting is indicative of ‘cold-start’ performance in a new environment (which is also the standard VLN evaluation setting in simulation).

As reported in row 8 of Table 2.2, in the ‘no map’ setting the instruction-following success rate drops a further 24.3% to 22.5%. This experiment also exhibits the lowest trajectory similarity with the results of the Matterport3D simulator (Table 2.3). Without the occupancy map, the robot collided with objects (a table and a chair) for the first time.

[2]Note that for pose tracking, we run a separate, isolated ROS navigation stack with map access.

However, we attribute most of the drop in success rate to differences between the predictions of the subgoal model and the locations of viewpoints in the simulator navigation graph. We found that both false positive and false negative waypoint predictions were evident, e.g., when the robot attempted to navigate down the spiral staircase (false positive), or failed to consider navigating through the narrow 'squeeze points' between bookshelves and other furniture (false negatives). Based on these results, we consider VLN sim-to-real transfer in this setting to be an open research problem.

	Platform	Camera	Adaptation	Map	4	5	6	7	8
4	Sim	Matterport	-	-	1.00	0.78	0.66	0.41	0.33
5	Sim	Theta V	✗	-	0.78	1.00	0.70	0.42	0.33
6	Sim	Theta V	✓	-	0.66	0.70	1.00	0.44	0.33
7	Robot	Theta V	✓	✓	0.41	0.42	0.44	1.00	0.57
8	Robot	Theta V	✓	✗	0.31	0.33	0.33	0.57	1.00

Table 2.3: Trajectory similarity between Coda experimental settings based on Normalized Dynamic Time Warping (**NDTW**) [80]. Figure 2.3 right provides an example to contextualize NDTW values.

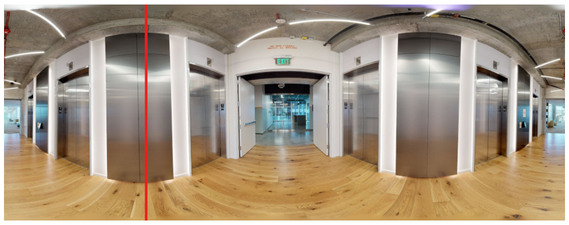
2.6 Conclusion and Future Directions

We attempt the first sim-to-real transfer of a Vision-and-Language Navigation (VLN) agent trained on the R2R dataset to a low-cost robot with 360° vision, using a learned subgoal model and classical SLAM and path-planning routines. We show that, if an occupancy map and navigation graph can be collected and annotated in advance, sim-to-real transfer is largely successful albeit with a $\sim$9% reduction in instruction following success due to visual domain differences ($\sim$4%) and viewpoint differences ($\sim$5%). This is a promising result, suggesting that the language groundings learned by the agent in simulation can transfer to a physical environment not seen in training. However, in the hardest 'cold start' setting with no prior mapping of the environment, sim-to-real transfer is much less reliable due to the failure of the subgoal model to predict the same neighboring waypoints in the simulator navigation

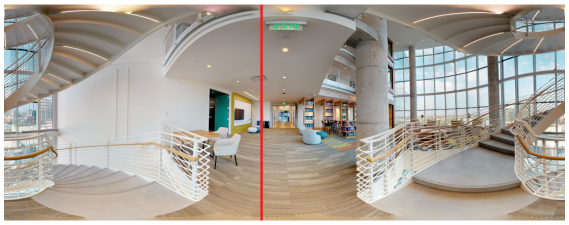
		Matterport3D / R2R Dataset					
		Train ($n = 61$)			Val-Unseen ($n = 11$)		
	Coda	Min	Avg	Max	Min	Avg	Max
Num Viewpoints	59	8	125	345	20	87	215
Navigation Graph Degree	3.6	2.2	4.0	5.4	3.1	3.8	4.9
Avg Edge Distance (m)	2.8	1.3	2.2	3.1	1.8	2.2	2.8
Num Instructions	111	6	230	300	18	214	300
Avg Instruction Length (words)	25	20	29	35	22	28	32
Avg Trajectory Length (m)	12.0	5.3	9.7	15.0	6.1	9.2	11.1
Avg Trajectory Edges	4.8	3.0	4.9	5.4	3.9	4.7	5.2

Table 2.4: Comparison of per-environment average statistics between Coda and R2R, suggesting that Coda is fairly typical of environments found in the Matterport3D / R2R dataset.

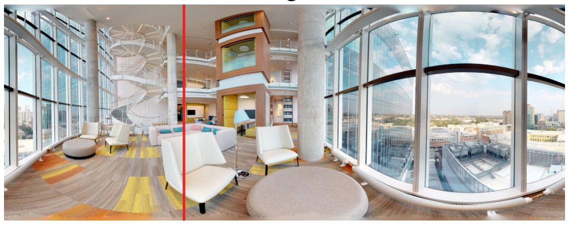
graph. To narrow the sim-to-real gap in future work, the subgoal model will need to be improved or eliminated, perhaps by training the VLN agent using a low-level action space (e.g., predicting the heading and distance to move). Since the graph-based Matterport3D simulator cannot support these low-level actions, this would require off-policy reinforcement learning algorithms that can learn from a fixed batch of data that has already been gathered [83], or alternatively, switching to a simulator that supports continuous motion [84, 85], as in recent work from [86]. Since these simulators introduce visual artifacts not present in the graph-based simulator, this may exacerbate visual domain differences, although recent work [87] demonstrates a promisingly small sim-to-real gap in the context of PointGoal navigation [88].



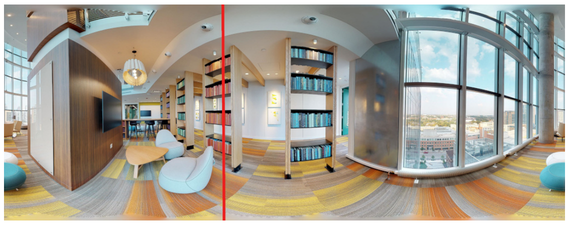
Turn right, move through the open open double doors. Turn right, move to the end of the hall. Turn right, wait in front of the drinking fountain.



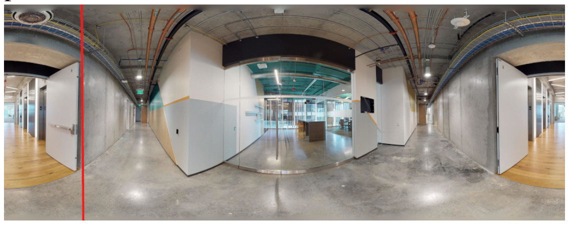
Walk straight towards the exit, but stop and make a right when you see the wall with art and stop.



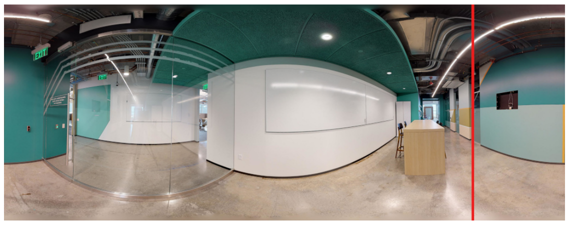
Make a right in front of the couch, veer straight, then right to proceed past the windows and stop between the first two bookshelves in front of the first hanging piece of art.



Go between the first set of bookshelves. Turn left and go straight until you are at the end of the hallway.



Turn around and walk to towards the end of the hall. At the intersection, turn and head into the men's room.



Continue forward with the whiteboards on your left. Keep walking and you will be in a new room. Stop before you reach the peach-colored couch on your right.

Figure 2.4: Additional examples of navigation instructions in the Coda environment. Each instruction is shown with the panoramic view from the starting pose, with the initial heading indicated in red.

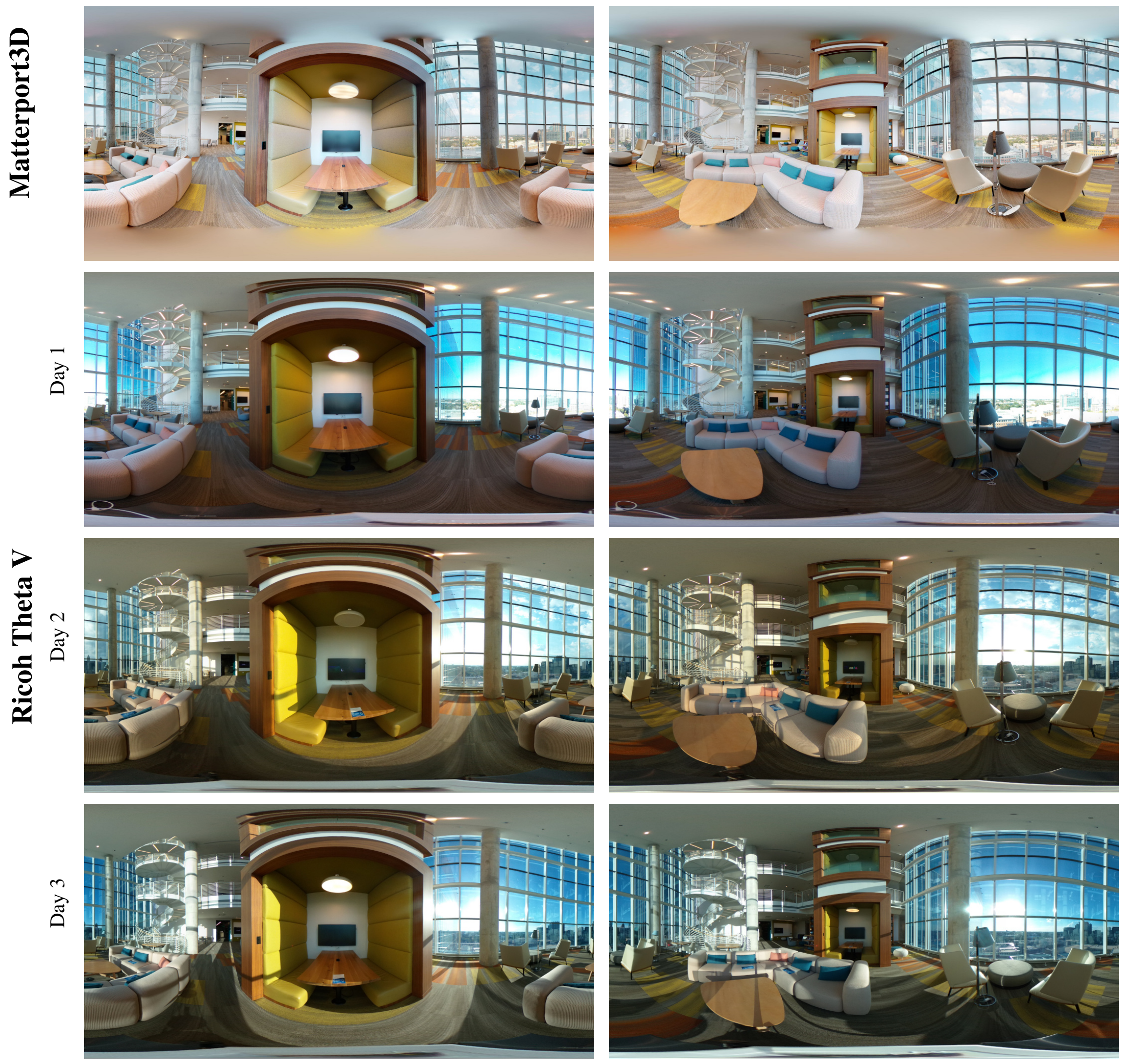


Figure 2.5: Panoramic captures in Coda from the Matterport3D camera (row 1) and the smaller and cheaper Ricoh Theta V camera (mounted on the robot) collected on three different days (rows 2-4). The robot camera's limited dynamic range and loss of detail as compared to the Matterport3D camera (used for training the VLN agent) is clearly evident. Images collected on different days (with the robot camera) illustrate the variations in shadows, lighting, and precise object placement that confront the robot in the real physical environment.

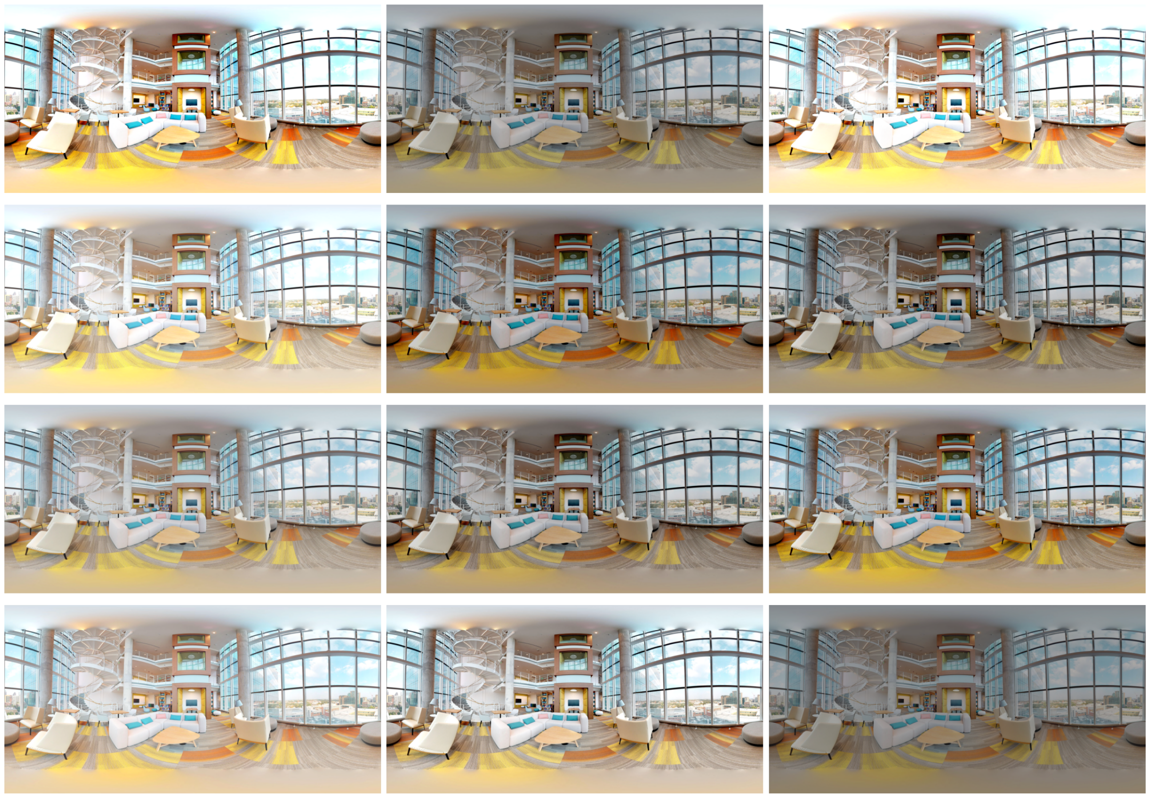


Figure 2.6: Examples of the random color jitter applied to each panorama while training the VLN agent to visually adapt to different lighting conditions.

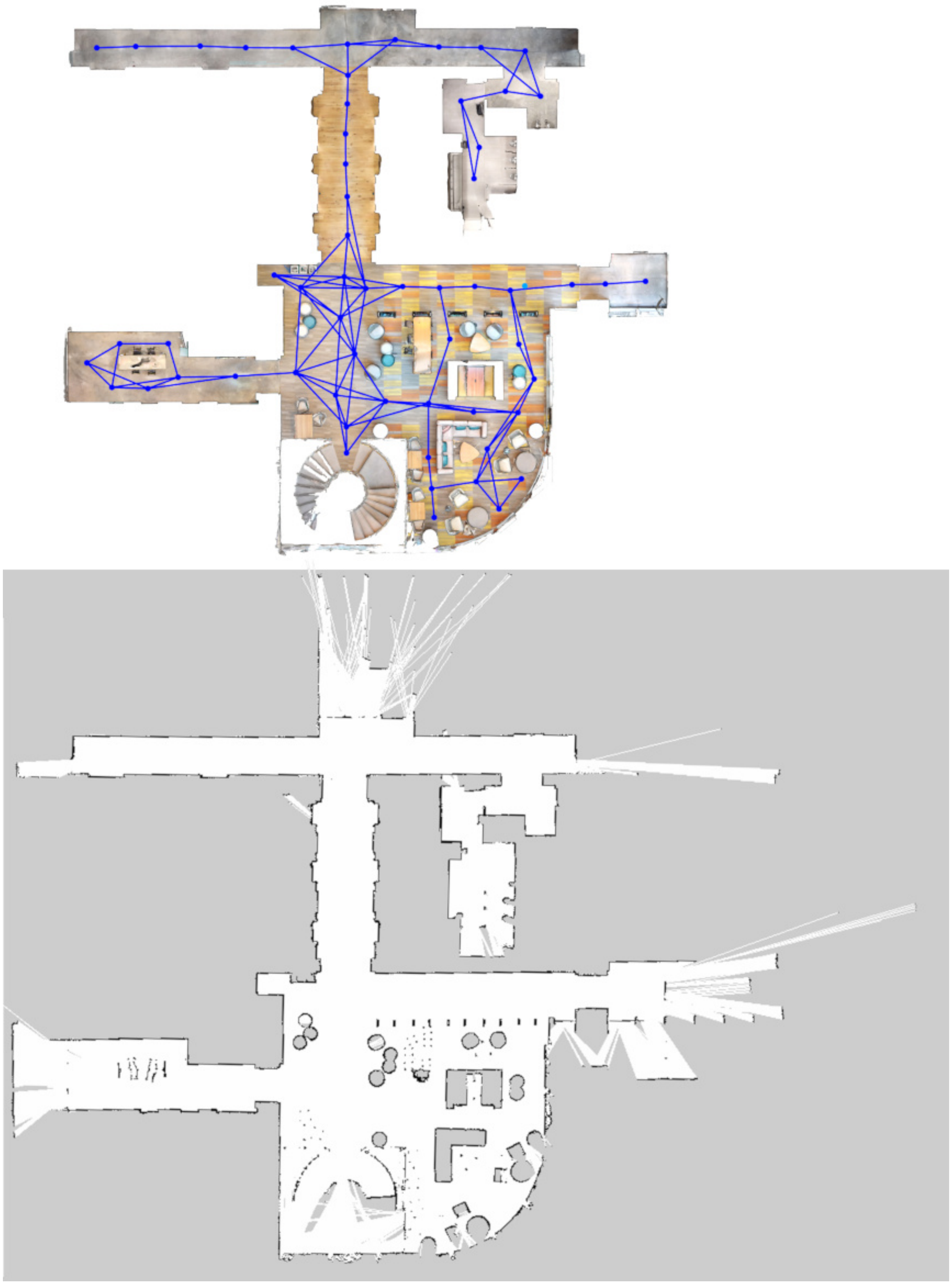


Figure 2.7: Floorplan view of Coda, showing the Matterport reconstruction and simulator navigation graph (top), and its close alignment to the 2D laser scan used for robot pose tracking (bottom).

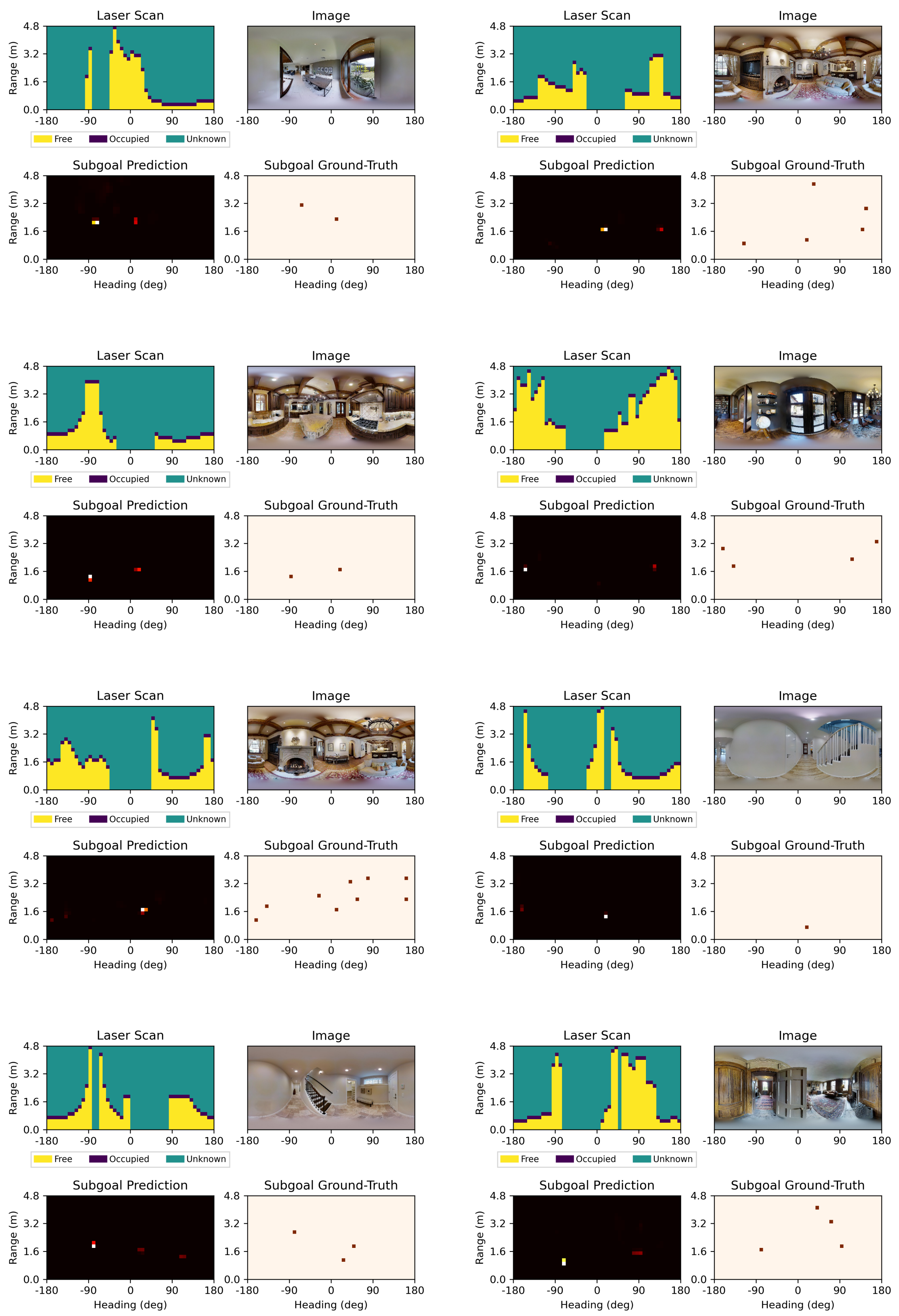
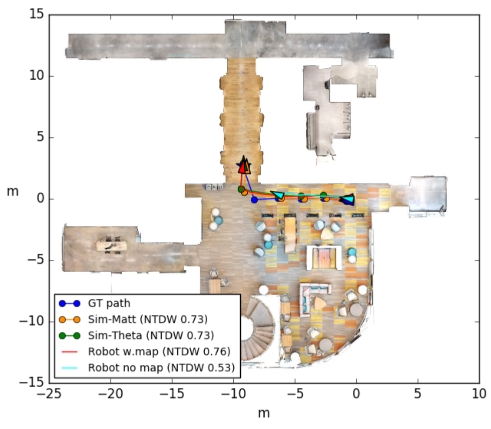
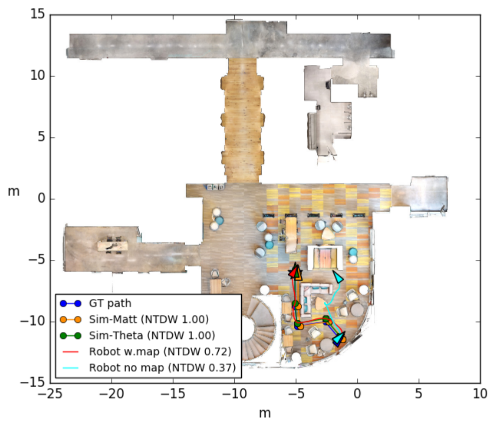


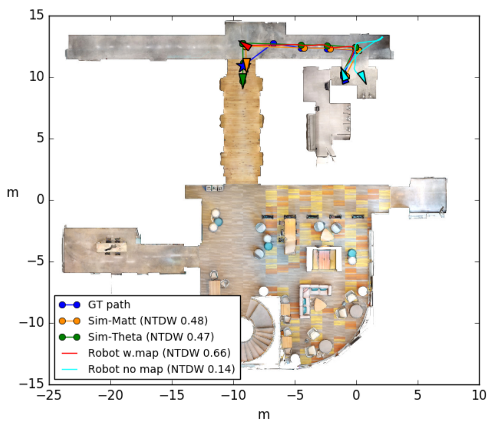
Figure 2.8: Waypoint predictions from the subgoal model on 8 randomly selected viewpoints from the Matterport validation set.



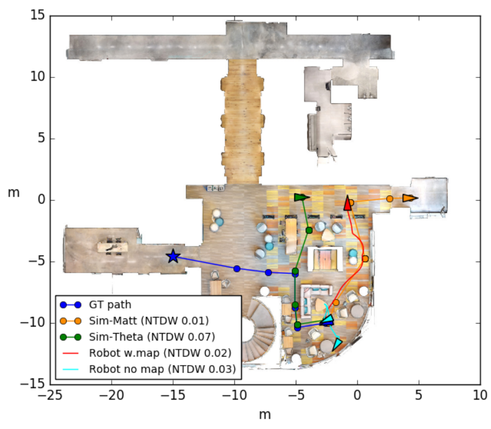
Instruction: Walk down the hall with the brown shelving units on your left. Turn right into the hallway with the elevators and then stop.



Instruction: Go between the table bookcase and the sectional sofa on this floor.

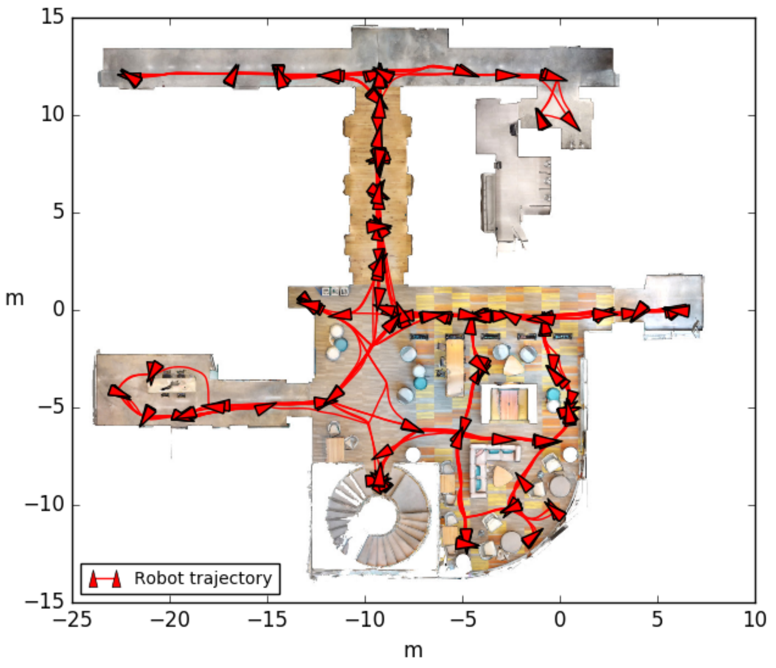


Instruction: Walk into hallway. Make a left at closed brown door. Walk down hall and make a left and stop by open white doors.



Instruction: Walk around the back of the large couch. Turn left towards the open green doorway. Walk through the green doors and stop.

Figure 2.9: Examples of Coda trajectories in sim and real for various instructions. While the robot's trajectory often resembles the simulator (top-left), subgoal prediction errors can lead to divergences between the 'with map' and 'no map' settings (top-right), particularly in areas of the building with floor-to-ceiling glass walls that are not easily detected (bottom-left). In the last example (bottom-right) the agent fails in both sim and real, highlighting the challenging nature of the VLN task.



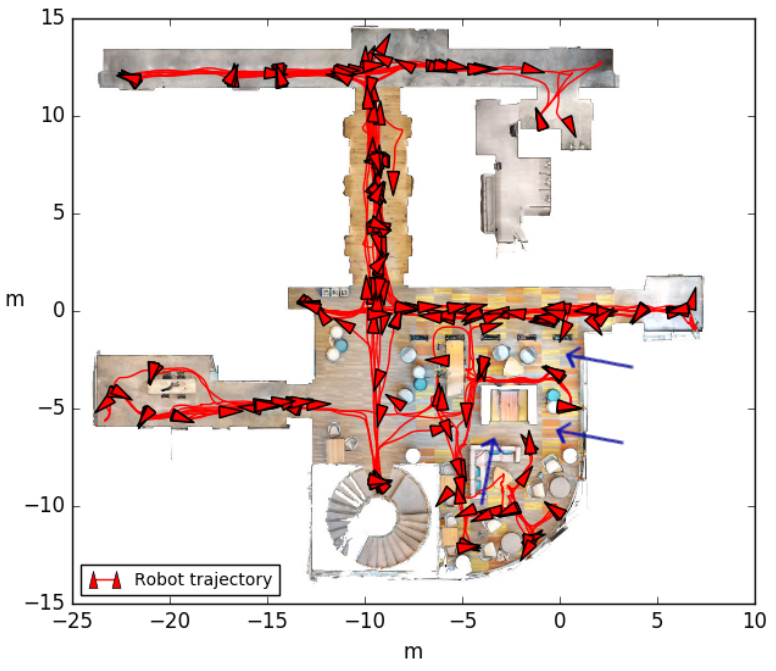
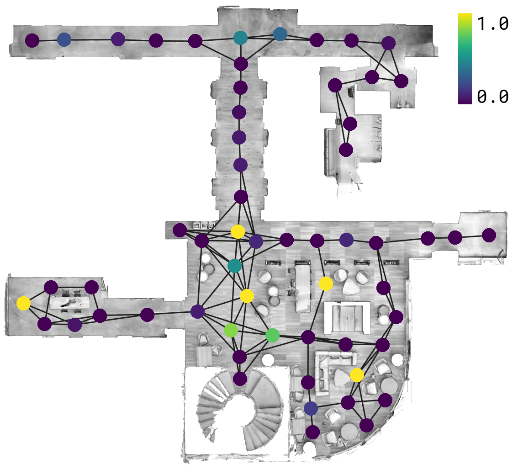
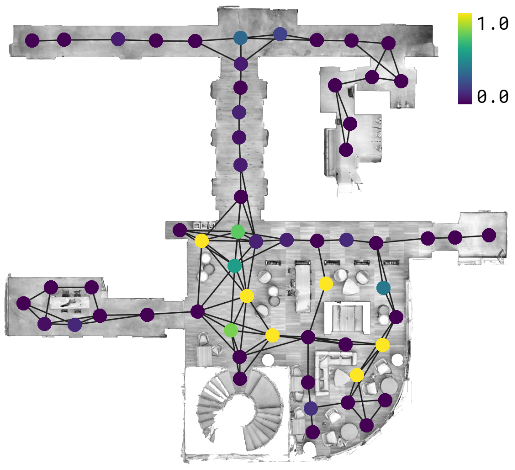


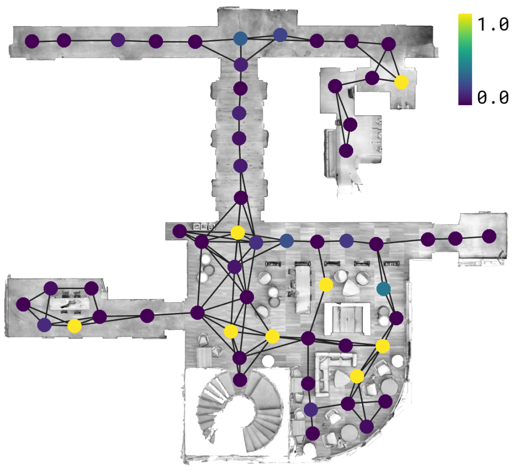
Figure 2.10: Illustration of all 111 of trajectories traversed by the robot under the ‘with map’ setting (top) and the ‘no map’ setting (bottom). With a map, the robot traversed the entire space without any collisions or navigation failures. Without a map, certain trajectory segments (highlighted) with blue arrows are never traversed, indicating that the subgoal model failed to predict these waypoints.



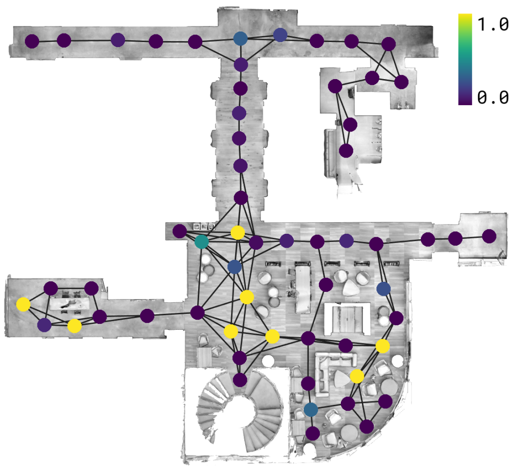
(a) Matterport3D



(b) Ricoh Theta V Day 1

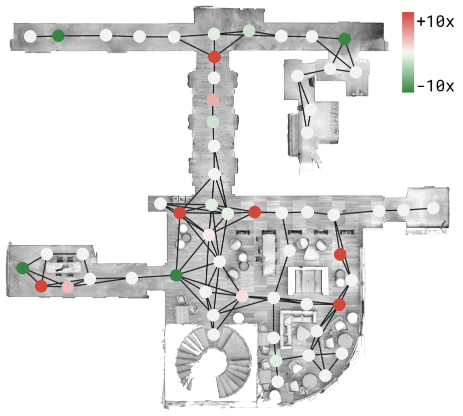


(c) Ricoh Theta V Day 2

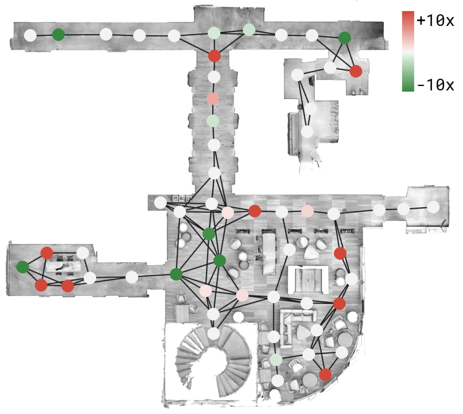


(d) Ricoh Theta V Day 3

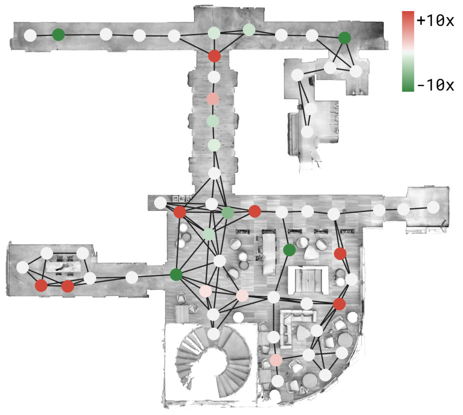
Figure 2.11: Illustration of the VLN agent's failure rates (in simulation) at each node in the navigation graph. The failure rates are consistently higher (yellow) in the bottom right of the map across all four data collections with the Matterport3D and Ricoh Theta V cameras.



(a) Failure Rate Ratio - Ricoh Theta V Day 1 to Matterport3D



(b) Failure Rate Ratio - Ricoh Theta V Day 2 to Matterport3D



(c) Failure Rate Ratio - Ricoh Theta V Day 3 to Matterport3D

Figure 2.12: Illustration of the log of the ratio between the failure rates with the Ricoh Theta V camera and the Matterport3D camera. A positive ratio, illustrated in red, indicates that the VLN agent was more likely to fail when processing data from the Ricoh Theta V camera. A negative ratio (in green) indicates the opposite. Across all three days, the agent was more likely to fail at nodes to the top and bottom of the elevator area and at nodes near the glass windows in the open space in the bottom right. Surprisingly, there are also some nodes (in green) at which the agent consistently performs better using the Ricoh Theta V panoramas.

Appendices

APPENDIX A
PUBLICATIONS

- **Chasing Ghosts: Instruction Following as Bayesian State Tracking**

  Peter Anderson*, Ayush Shrivastava*, Devi Parikh, Dhruv Batra, Stefan Lee

  Advances in Neural Information Processing Systems (NeurIPS) 2019

- **Improving Vision-and-Language Navigation with Image-Text Pairs from the Web**

  Arjun Majumdar, Ayush Shrivastava, Stefan Lee, Peter Anderson, Devi Parikh, Dhruv Batra

  European Conference on Computer Vision (ECCV) 2020

- **Sim-to-Real Transfer for Vision-and-Language Navigation**

  Peter Anderson, Ayush Shrivastava, Joanne Truong, Arjun Majumdar, Devi Parikh, Dhruv Batra, Stefan Lee

  Conference on Robot Learning (CoRL) 2020

REFERENCES

[1] 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.

[2] D. Misra, A. Bennett, V. Blukis, E. Niklasson, M. Shatkhin, and Y. Artzi, "Mapping instructions to actions in 3d environments with visual goal prediction," in *EMNLP*, 2018.

[3] X. Wang, Q. Huang, A. Celikyilmaz, J. Gao, D. Shen, Y.-F. Wang, W. Y. Wang, and L. Zhang, "Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation," in *CVPR*, 2019.

[4] X. Wang, W. Xiong, H. Wang, and W. Y. Wang, "Look before you leap: Bridging model-free and model-based reinforcement learning for planned-ahead vision-and-language navigation," in *ECCV*, 2018.

[5] C.-Y. Ma, J. Lu, Z. Wu, G. AlRegib, Z. Kira, R. Socher, and C. Xiong, "Self-monitoring navigation agent via auxiliary progress estimation," in *ICLR*, 2019.

[6] C.-Y. Ma, Z. Wu, G. AlRegib, C. Xiong, and Z. Kira, "The regretful agent: Heuristic-aided navigation through progress estimation," in *CVPR*, 2019.

[7] D. Fried, R. Hu, V. Cirik, A. Rohrbach, J. Andreas, L.-P. Morency, T. Berg-Kirkpatrick, K. Saenko, D. Klein, and T. Darrell, "Speaker-follower models for vision-and-language navigation," in *NeurIPS*, 2018.

[8] H. Tan, L. Yu, and M. Bansal, "Learning to navigate unseen environments: Back translation with environmental dropout," in *NAACL*, 2019.

[9] L. Ke, X. Li, Y. Bisk, A. Holtzman, Z. Gan, J. Liu, J. Gao, Y. Choi, and S. Srinivasa, "Tactical rewind: Self-correction via backtracking in vision-and-language navigation," in *CVPR*, 2019.

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

[11] V. Blukis, D. Misra, R. A. Knepper, and Y. Artzi, "Mapping navigation instructions to continuous control actions with position-visitation prediction," in *CoRL*, 2018.

[12] J. F. Henriques and A. Vedaldi, "Mapnet: An allocentric spatial memory for mapping environments," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[13] D. Gordon, A. Kembhavi, M. Rastegari, J. Redmon, D. Fox, and A. Farhadi, "IQA: Visual question answering in interactive environments," in *CVPR*, 2018.

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

[15] R. Jonschkowski and O. Brock, "End-to-end learnable histogram filters," in *In Workshop on Deep Learning for Action and Interaction at the Conference on Neural Information Processing Systems (NIPS)*, 2016.

[16] P. Anderson, A. 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:1807.06757*, 2018.

[17] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3d: Learning from rgb-d data in indoor environments," *International Conference on 3D Vision (3DV)*, 2017.

[18] T. Winograd, "Procedures as a representation for data in a computer program for understanding natural language," Massachusetts Institute of Technology, Tech. Rep., 1971.

[19] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, 1997.

[20] D. Wierstra, A. Foerster, J. Peters, and J. Schmidhuber, "Solving deep memory pomdps with recurrent policy gradients," in *International Conference on Artificial Neural Networks*, 2007.

[21] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," in *ICLR*, 2017.

[22] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, *et al.*, "Learning to navigate in complex environments," in *ICLR*, 2017.

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

[24] A. Das, S. Datta, G. Gkioxari, S. Lee, D. Parikh, and D. Batra, "Embodied Question Answering," in *CVPR*, 2018.

[25] J. Oh, V. Chockalingam, S. Singh, and H. Lee, "Control of memory, active perception, and action in minecraft," in *ICML*, 2016.

[26] E. Parisotto and R. Salakhutdinov, "Neural map: Structured memory for deep reinforcement learning," in *ICLR*, 2018.

[27] N. Savinov, A. Dosovitskiy, and V. Koltun, "Semi-parametric topological memory for navigation," in *ICLR*, 2018.

[28] T. Haarnoja, A. Ajay, S. Levine, and P. Abbeel, "Backprop KF: Learning discriminative deterministic state estimators," in *NIPS*, 2016.

[29] R. Jonschkowski, D. Rastogi, and O. Brock, "Differentiable Particle Filters: End-to-End Learning with Algorithmic Priors," in *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

[30] P. Karkus, D. Hsu, and W. S. Lee, "Particle filter networks with application to visual localization," in *Proceedings of the Annual Conference on Robot Learning (CoRL)*, 2018.

[31] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *NIPS*, 2015.

[32] 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.

[33] J. Uhrig, N. Schneider, L. Schneider, U. Franke, T. Brox, and A. Geiger, "Sparsity invariant CNNs," in *2017 International Conference on 3D Vision (3DV)*, IEEE, 2017, pp. 11–20.

[34] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *ICLR*, 2015.

[35] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *EMNLP*, 2014.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[37] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, 2015.

[38] P. K. Nathan Silberman Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.

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

[40] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *IJCV*, 2015.

[41] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, "Efficient object localization using convolutional networks," in *CVPR*, 2015.

[42] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, 2014.

[43] S. Guadarrama, L. Riano, D. Golland, D. Go, Y. Jia, D. Klein, P. Abbeel, T. Darrell, *et al.*, "Grounding spatial relations for human-robot interaction," in *IROS*, 2013.

[44] D. K. Misra, J. Sung, K. Lee, and A. Saxena, "Tell me dave: Context-sensitive grounding of natural language to manipulation instructions," *The International Journal of Robotics Research*, vol. 35, no. 1-3, pp. 281–300, 2016.

[45] R. Paul, J. Arkin, N. Roy, and T. M Howard, "Efficient grounding of abstract spatial concepts for natural language interaction with robot manipulators," *RSS*, 2016.

[46] D. L. Chen and R. J. Mooney, "Learning to interpret natural language navigation instructions from observations.," in *AAAI*, 2011.

[47] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. J. Teller, and N. Roy, "Understanding natural language commands for robotic navigation and mobile manipulation.," in *AAAI*, 2011.

[48] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox, "Learning to parse natural language commands to a robot control system," in *Experimental Robotics*, Springer, 2013, pp. 403–415.

[49] Y. Bisk, D. Yuret, and D. Marcu, "Natural language communication with robots," in *NAACL-HLT*, 2016.

[50] D. S. Chaplot, K. M. Sathyendra, R. K. Pasumarthi, D. Rajagopal, and R. Salakhutdinov, "Gated-attention architectures for task-oriented language grounding," *arXiv:1706.07230*, 2017.

[51] T. Kollar, S. Tellex, D. Roy, and N. Roy, "Toward understanding natural language directions," in *Human-Robot Interaction (HRI)*, 2010.

[52] M. MacMahon, B. Stankiewicz, and B. Kuipers, “Walk the talk: Connecting language, knowledge, and action in route instructions,” in *AAAI*, 2006.

[53] H. Mei, M. Bansal, and M. R. Walter, “Listen, attend, and walk: Neural mapping of navigational instructions to action sequences.,” in *AAAI*, 2016.

[54] A. Vogel and D. Jurafsky, “Learning to follow navigational directions,” in *ACL*, 2010.

[55] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: An open-source robot operating system,” in *ICRA workshop on open source software*, vol. 3, 2009, p. 5.

[56] G. Grisetti, C. Stachniss, and W. Burgard, “Improved techniques for grid mapping with rao-blackwellized particle filters,” *IEEE Transactions on Robotics*, 2007.

[57] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world,” in *IROS*, 2017.

[58] S. Hemachandra, F. Duvallet, T. M. Howard, N. Roy, A. Stentz, and M. R. Walter, “Learning models for following natural language directions in unknown environments,” in *ICRA*, 2015.

[59] A. S. Huang, S. Tellex, A. Bachrach, T. Kollar, D. Roy, and N. Roy, “Natural language command of an autonomous micro-air vehicle,” in *IROS*, 2010.

[60] V. Blukis, Y. Terme, E. Niklasson, R. A. Knepper, and Y. Artzi, “Learning to map natural language instructions to physical quadcopter control using simulated flight,” in *CoRL*, 2019.

[61] J. Oh, T. M. Howard, M. R. Walter, D. Barber, M. Zhu, S. Park, A. Suppe, L. Navarro-Serment, F. Duvallet, A. Boularias, *et al.*, “Integrated intelligence for human-robot teams,” in *International Symposium on Experimental Robotics*, Springer, 2016, pp. 309–322.

[62] F. Codevilla, M. Miiller, A. López, V. Koltun, and A. Dosovitskiy, “End-to-end driving via conditional imitation learning,” in *ICRA*, 2018.

[63] S. Patki, E. Fahnestock, T. M. Howard, and M. R. Walter, “Language-guided semantic mapping and mobile manipulation in partially observable environments,” in *CoRL*, 2019.

[64] M. Tucker, D. Aksaray, R. Paul, G. J. Stein, and N. Roy, “Learning unknown groundings for natural language interaction with mobile robots,” in *Robotics Research*, Springer, 2020, pp. 317–333.

[65] S. Banerjee, J. Thomason, and J. J. Corso, "The RobotSlang Benchmark: Dialog-guided robot localization and navigation," in *CORL*, 2020.

[66] E. Fahnestock, S. Patki, and T. M. Howard, "Language-guided adaptive perception with hierarchical symbolic representations for mobile manipulators," *arXiv:1909.09880*, 2019.

[67] R. Paul, A. Barbu, S. Felshin, B. Katz, and N. Roy, "Temporal grounding graphs for language understanding with accrued visual-linguistic context," *arXiv:1811.06966*, 2018.

[68] J. Thomason, M. Murray, M. Cakmak, and L. Zettlemoyer, "Vision-and-dialog navigation," in *CoRL*, 2019.

[69] K. Nguyen and H. Daumé III, "Help, Anna! visual navigation with natural multimodal assistance via retrospective curiosity-encouraging imitation learning," in *EMNLP*, 2019.

[70] Y. Qi, Q. Wu, P. Anderson, X. Wang, W. Y. Wang, C. Shen, and A. van den Hengel, "Reverie: Remote embodied visual referring expression in real indoor environments," in *CVPR*, 2020.

[71] A. Ku, P. Anderson, R. Patel, E. Ie, and J. Baldridge, "Room-Across-Room: Multilingual vision-and-language navigation with dense spatiotemporal grounding," in *EMNLP*, 2020.

[72] A. Yan, X. Wang, J. Feng, L. Li, and W. Y. Wang, "Cross-lingual vision-language navigation," *arXiv:1910.11301*, 2019.

[73] P. Mirowski, M. Grimes, M. Malinowski, K. M. Hermann, K. Anderson, D. Teplyashin, K. Simonyan, A. Zisserman, R. Hadsell, *et al.*, "Learning to navigate in cities without a map," in *NeurIPS*, 2018.

[74] H. Mehta, Y. Artzi, J. Baldridge, E. Ie, and P. Mirowski, "Retouchdown: Adding touchdown to streetlearn as a shareable resource for language grounding tasks in street view," *EMNLP Workshop on Spatial Language Understanding (SpLU)*, 2020.

[75] M. Weiss, S. Chamorro, R. Girgis, M. Luck, S. E. Kahou, J. P. Cohen, D. Nowrouzezahrai, D. Precup, F. Golemo, and C. Pal, "Navigation agents for the visually impaired: A sidewalk simulator and experiments," *arXiv:1910.13249*, 2019.

[76] H. Chen, A. Suhr, D. Misra, and Y. Artzi, "Touchdown: Natural language navigation and spatial reasoning in visual street environments," in *CVPR*, 2019.

[77] A. B. Vasudevan, D. Dai, and L. Van Gool, “Talk2nav: Long-range vision-and-language navigation in cities,” *arXiv:1910.02029*, 2019.

[78] T. Paz-Argaman and R. Tsarfaty, “Run through the streets: A new dataset and baseline models for realistic urban navigation,” *arXiv:1909.08970*, 2019.

[79] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in PyTorch,” in *NIPS Autodiff Workshop*, 2017.

[80] G. Magalhaes, V. Jain, A. Ku, E. Ie, and J. Baldridge, “Effective and general evaluation for instruction conditioned navigation using dynamic time warping,” *arXiv:1907.05446*, 2019.

[81] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *CVPR*, 2016.

[82] M. Cuturi, “Sinkhorn distances: Lightspeed computation of optimal transport,” in *NIPS*, 2013.

[83] S. Fujimoto, D. Meger, and D. Precup, “Off-policy deep reinforcement learning without exploration,” in *ICML*, 2019.

[84] 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.

[85] Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, “Habitat: A Platform for Embodied AI Research,” in *ICCV*, 2019.

[86] 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.

[87] A. Kadian, J. Truong, A. Gokaslan, A. Clegg, E. Wijmans, S. Lee, M. Savva, S. Chernova, and D. Batra, “Are we making real progress in simulated environments? Measuring the sim2real gap in embodied visual navigation,” in *IROS*, 2020.

[88] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, *et al.*, “On evaluation of embodied navigation agents,” *arXiv:1807.06757*, 2018.