Mapping Navigation Instructions to Continuous Control Actions with Position-Visitation Prediction

Valts Blukis[†◇] **Dipendra Misra**[†◇] **Ross A. Knepper**[†] **Yoav Artzi**[†◇]

[†]Department of Computer Science, Cornell University, Ithaca, New York, USA
[◇]Cornell Tech, Cornell University, New York, New York, USA
{valts, dkm, rak, yoav}@cs.cornell.edu

Abstract: We propose an approach for mapping natural language instructions and raw observations to continuous control of a quadcopter drone. Our model predicts interpretable position-visitation distributions indicating where the agent should go during execution and where it should stop, and uses the predicted distributions to select the actions to execute. This two-step model decomposition allows for simple and efficient training using a combination of supervised learning and imitation learning. We evaluate our approach with a realistic drone simulator, and demonstrate absolute task-completion accuracy improvements of 16.85% over two state-of-the-art instruction-following methods.

Keywords: Natural language understanding; Quadcopter; Simulation; Instruction Following; Imitation Learning

1 Introduction

Executing natural language navigation instructions from raw observations requires solving language, perception, planning, and control problems. Consider instructing a quadcopter drone using natural language. Figure 1 shows an example instruction. Resolving the instruction requires identifying the *blue fence*, *anvil* and *tree* in the world, understanding the spatial constraints *towards* and *on the right*, planning a trajectory that satisfies these constraints, and continuously controlling the quadcopter to follow the trajectory. Existing work has addressed this problem mostly using manually-designed symbolic representations for language meaning and environment [1, 2, 3, 4, 5, 6]. This approach requires significant knowledge representation effort and is hard to scale. Recently, Blukis et al. [7] proposed to trade-off the representation design with representation learning. However, their approach was developed using synthetic language only, where a small set of words were combined in a handful of different ways. This does not convey the full complexity of natural language, and may lead to design decisions and performance estimates divorced from the real problem.

In this paper, we study the problem of executing instructions with a realistic quadcopter simulator using a corpus of crowdsourced natural language navigation instructions. Our data and environment combine language and robotic challenges. The instruction language is rich with linguistic phenomena, including object references, co-references within sentences, and spatial and temporal relations; the environment simulator provides a close approximation of realistic quadcopter flight, including a realistic controller that requires rapid decisions in response to continuously changing observations.

We address the complete execution problem with a single model that is decomposed into two stages of planning and plan execution. Figure 2 illustrates the two stages. The first stage takes as input the language and the observations of the agent, and outputs two distributions that aim to to solve different challenges: (a) identifying the positions that are likely to be visited during a correct instruction execution, and (b) recognizing the correct goal position. The second stage of the model controls the drone to fly between the high probability positions to complete the task and reach the most likely goal location. The two stages are combined into a single neural network. While the approach does not require designing an intermediate symbolic representation, the agent plan is still interpretable by simple visualization of the distributions over a map.

Our approach introduces two learning challenges: (a) estimate the model parameters with the limited language data available and a realistic number of experiences in the environment; and (b) ensure the

2nd Conference on Robot Learning (CoRL 2018), Zürich, Switzerland.

Go towards the blue fence passing the anvil and tree on the right

Figure 1: An example task from the LANI dataset, showing the environment, agent, and the natural language instruction.

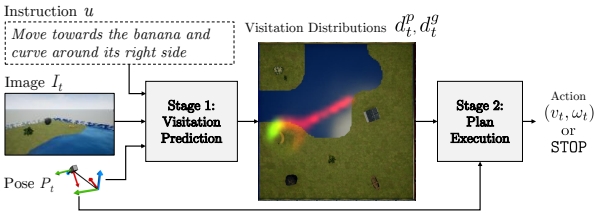

Figure 2: High-level illustration of our model. The model is decomposed into a visitation prediction component that predicts which areas in the environment to visit, and an execution component that outputs actions to drive the agent through predicted high-likelihood regions. The visitation prediction component predicts the positions to visit (red) and where to stop (green).

different parts of the model specialize to solve their intended tasks. We address both challenges by training each of the two stages separately. We train the visitation prediction stage with supervised learning using expert demonstrations, and the plan execution stage by mapping expert visitation distributions to actions using imitation learning. At test time, the second stage uses the predicted distributions from the first. This learning method emphasizes sample efficiency. The first stage uses supervised learning with training demonstrations; the second stage is independent from the complex language and visual reasoning required, allowing for sample-efficient imitation learning. This approach also does not suffer from the credit assignment problem of training the complete network using rewards on actions only. This ensures the different parts of the network solve their intended task, and the generated interpretable distributions are representative of the agent reasoning.

To evaluate our approach, we adapt the LANI corpus [8] for the realistic quadcopter simulator from Blukis et al. [7], and create a continuous control instruction following benchmark. The LANI corpus includes 27,965 crowdsourced natural language instructions paired with human demonstrations. We compare our approach to the continuous-action analogs of two recently proposed approaches [9, 10], and demonstrate absolute task-completion accuracy improvements of 16.85%. We also discuss a generalization of our position visitation prediction approach to state-visitation distribution prediction for sequential decision processes, and suggest the conditions for applying it to future work on robot learning problems. The models, dataset, and environment are publicly available at `https://github.com/clic-lab/drif`.

2 Technical Overview

Task Let $\mathcal{U}$ be the set of natural language instructions, $\mathcal{S}$ be the set of world states, and $\mathcal{A}$ be the set of all actions. An instruction u is a sequence of l tokens $\langle u_1, \ldots, u_l \rangle$. An action a is either a tuple (v, ω) of forward and angular velocities or the completion action `STOP`. The state s contains information about the current configuration of all objects in the world. Given a start state $s_1 \in \mathcal{S}$ and an instruction $u \in \mathcal{U}$, the agent executes u by generating a sequence of actions, where the last action is the special action `STOP`, which indicates task completion. The agent behavior is determined by its configuration ρ. An execution of length T is a sequence $\langle (s_1, a_1), \ldots, (s_T, a_T) \rangle$, where $s_t \in \mathcal{S}$ is the state at timestep t, $a_t \in \mathcal{A}$ is the action updating the agent configuration, and the last action is $a_T = \texttt{STOP}$. Given an action $a_t = (v_t, \omega_t)$, we set the agent configuration $\rho = (v_t, \omega_t)$, which specifies the controller setpoint. Between actions, the agent maintains its configuration.

Model The agent does not have access to the world state. At timestep t, the agent observes the agent context $c_t = (u, I_1, \cdots, I_t, P_1, \cdots P_t)$, where u is the instruction and $I_i = \text{IMG}(s_i)$ and $P_i = \text{LOCALIZE}(s_i)$, $i = 1 \ldots t$ are monocular first-person RGB images and 6-DOF agent poses observed at time step i. The pose P_i is a pair (p_i, γ_i), where p_i is a position and γ_i is an orientation. Given an agent context c_t, we predict two visitation distributions that define a plan to execute and the actions required to execute the plan. A visitation distribution is a discrete distribution over positions in the environment. The trajectory-visitation distribution d^p puts high probability on positions in the environment the agent is likely to go through during execution, and the goal-visitation distribution d^g puts high probability on positions where the agent should `STOP` to complete its execution. Given d^p and d^g, the second stage of the model predicts the actions to complete the task by going through high

probability positions according to the distributions. As the agent observes more of the environment during execution, the distributions are continuously updated.

Learning We assume access to a training set of N examples $\{(u^{(i)}, s_1^{(i)}, \Xi^{(i)})\}_{i=1}^N$, where $u^{(i)}$ is an instruction, $s_1^{(i)}$ is a start state, and $\Xi^{(i)} = \langle p_1^{(i)}, \ldots, p_T^{(i)} \rangle$ is a sequence of positions that defines a trajectory generated from a human demonstration execution of u. Learning is decomposed into two stages. We first train the visitation distributions prediction given the visitation distributions inferred from the oracle policy π^*. We then use imitation learning using π^* to generate the sequence of actions required given the visitation distributions.

Evaluation We evaluate on a test set of M examples $\{(u^{(i)}, s_1^{(i)}, p_g^{(i)})\}_{i=1}^M$, where $u^{(i)}$ is an instruction, $s_1^{(i)}$ is a start state, and $p_g^{(i)}$ is the goal position. We consider the task successfully completed if the agent outputs the `STOP` action within a predefined Euclidean distance of $p_g^{(i)}$. We additionally evaluate the mean and median Euclidean distance to the goal position.

3 Related Work

Natural language instruction following has been studied extensively on physical robots [11, 12, 13, 2, 14, 12, 15, 16] and simulated agents [17, 18, 19, 3, 20, 21]. These methods require hand-engineering of intermediate symbolic representations, an effort that is hard to scale to complex domains. Our approach does not require a symbolic representation, instead relying on a learned spatial representation induced directly from human demonstrations. Our approach is related to recent work on executing instructions without such symbolic representations using discrete environments [9, 22, 23, 8]. In contrast, we use a continuous environment. While we focus on the challenge of using natural language, this problem was also studied using synthetic language with the goal of abstracting natural language challenges and focusing on navigation [24, 10] and continuous control [7].

Our approach is related to recent work on learning visuomotor control policies for grasping [25, 26, 27], dexterous manipulation [28, 29, 30] and visual navigation [31]. While these methods have mostly focused on learning single robotic tasks, or transferring a single task between multiple domains [30, 32, 33, 34], our aim is to train a model that can execute navigation tasks specified using natural language, including previously unseen tasks during test time.

Treating planning as prediction of visitation probabilities is related to recent work on neural network models that explicitly construct internal maps [35, 7, 36], incorporate external maps [31, 37], or do planning [32]. These architectures take advantage of domain knowledge to provide sample-efficient training and interpretable representations. In contrast, we cast planning as an image-to-image mapping [38, 39], where the output image is interpreted as a probability distribution over environment locations. Our architecture borrows building blocks from prior work. We use the ResNet architecture for perception [40] and the neural mapping approach of Blukis et al. [7] to construct a dynamic semantic map. We also use the LINGUNET conditional image translation module [8]. While it was introduced for first-person goal location prediction, we use it to predict visitation distributions.

Learning from Demonstrations (LfD) approaches have previously decomposed robot learning into learning high-level tasks and low-level skills (e.g. Dynamic Movement Primitives [41, 42, 43, 44]). Our approach follows this general idea. However, instead of using trajectories or probabilities as task representations [45], we predict visitation distributions using a neural network. This results in a reactive approach that defers planning of the full trajectory and starts task execution under uncertainty that is gradually reduced with additional observations. This approach does not assume access to the full system configuration space or a symbolic environment representation. Furthermore, the learned representation is not constrained to a specific robot. For example, the same predicted visitation distribution could potentially be used on a humanoid or a ground vehicle, each running its own plan execution component.

4 Model

We model the agent behavior using a neural network policy π. The input to the policy at time t is the agent context $c_t = (u, I_1, \cdots, I_t, P_1, \cdots P_t)$, where u is the instruction and $I_i = \textsc{Img}(s_i)$ and $P_i = \textsc{Localize}(s_i)$, $i = 1 \ldots t$ are first-person images and 6-DOF agent poses observed at timestep i and state s_i. The policy outputs an action $a_t = (v_t, \omega_t)$, where v_t is a forward velocity

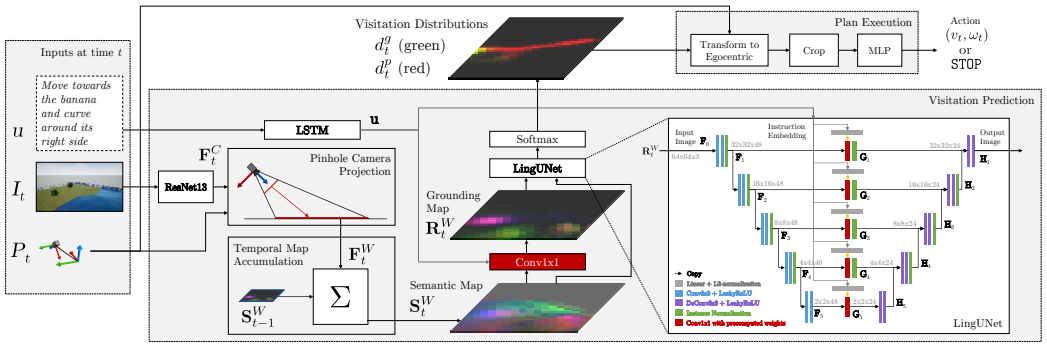

Figure 3: An illustration of our model architecture. The instruction u is encoded into an instruction embedding $\mathbf{u}$ using an LSTM network. At each timestep t, image features $\mathbf{F}_t^C$ are produced with a custom residual network [ResNet; 40], projected to the world reference frame through a pinhole camera model, and accumulated through time into a globally persistent semantic map $\mathbf{S}_t^W$. The map is used to predict the visitation distributions d^p and d^g by using $\mathbf{u}$ to create a grounding map $\mathbf{R}_t^W$ and generate the distributions using the LINGUNET architecture. A simple execution network then transforms the distributions to an egocentric reference frame and generates the next action.

Figure 4: Predicted visitation distributions for the the instruction from the LANI development set *go around the barrel and then move towards the phone booth*. The left-most image shows top-down view with the human demonstration trajectory (red) and our model trajectory (blue). The next four images are predicted visitation distributions, d^p (red) and d^g (green), as the execution progresses (left to right). The white circle represents agent's current position. The uncertainty over the stopping position decreases with time as the semantic map accumulates more information. The right image shows the first three channels of the final semantic map $\mathbf{S}_T^W$.

and ω_t is an angular velocity, and a probability for the STOP action p_t^{stop}. We decompose the policy to visitation prediction and plan execution. Visitation prediction $\text{VISIT}(c_t)$ computes a 2D discrete semantic map $\mathbf{S}_t^W$. Each position in $\mathbf{S}_t^W$ corresponds to an area in the environment, and represents it with a learned vector. The map is used to generate two probability distributions: trajectory-visitation distribution $d_t^p(\tilde{p} \mid c_t)$ and goal-visitation distribution $d_t^g(\tilde{p} \mid c_t)$, where $\tilde{p}$ is a position in $\mathbf{S}_t^W$. The first distribution models the probability of visiting each position as part of an optimal policy executing the instruction u, and the second the probability of each position being the goal where the agent should select the STOP action. We update the semantic map $\mathbf{S}_t^W$ at every timestep with the latest observations. The distributions d_t^p and d_t^g are only computed every T_d timesteps. When not updating the distributions, we set $d_t^p = d_{t-1}^p$ and $d_t^g = d_{t-1}^g$. This allows for periodic re-planning and limits the computational workload. In the second stage, plan execution $\text{ACT}(d_t^p, d_t^g, P_t)$ generates the action a_t and the stop probability p_t^{stop}. Figure 3 illustrates our architecture, and Figure 4 shows example visitation distributions generated by our approach.

4.1 Stage 1: Visitation Prediction

Feature Projection and Semantic Mapping We predict the visitation distributions over a learned semantic map of the environment. We construct the map using the method of Blukis et al. [7]. The full details of the process are specified in the original paper. Roughly speaking, the semantic mapping process includes three steps: feature extraction, projection, and accumulation. At timestep t, we process the currently observed image I_t using a 13-layer residual neural network RESNET to generate a feature map $\mathbf{F}_t^C = \text{RESNET}(I_t)$ of size $W_f \times H_f \times C$. We compute a feature map in the world coordinate frame $\mathbf{F}_t^W$ by projecting $\mathbf{F}_t^C$ with a pinhole camera model onto the ground plane at elevation zero. The semantic map of the environment $\mathbf{S}_t^W$ at time t is an integration of $\mathbf{F}_t^W$ and $\mathbf{S}_{t-1}^W$, the map from the previous timestep. The integration equation is given in Section 4c in Blukis et al. [7]. This process generates a tensor $\mathbf{S}_t^W$ of size $W_w \times H_w \times C$ that represents a map, where each location $[\mathbf{S}_t^W]_{(x,y)}$ is a C-dimensional feature vector computed from all past observations $I_{<t}$,

each processed to learned features $\mathbf{F}^C_{\leq t}$ and projected onto the environment ground in the world frame at coordinates (x, y). This map maintains a learned high-level representation for every world location (x, y) that has been visible in any of the previously observed images. We define the world coordinate frame using the agent starting pose P_1: the agent position is the coordinates $(0, 0)$, and the positive direction of the x-axis is along the agent heading. This gives consistent meaning to spatial language, such as *turn left* or *pass on the left side of*.

Instruction Embedding We represent the instruction $u = \langle u_1, \cdots u_l \rangle$ as an embedded vector $\mathbf{u}$. We generate a series of hidden states $\mathbf{h}_i = \text{LSTM}(\phi(u_i), \mathbf{h}_{i-1})$, $i = 1 \dots l$, where LSTM is a Long-Short Term Memory [46] Recurrent Neural Network (RNN) and ϕ is a learned word-embedding function. The instruction embedding is the last hidden state $\mathbf{u} = \mathbf{h}_l$.

Position Visitation Distribution Prediction We use image generation to predict the visitation distributions d^p_t and d^g_t. For each of the two distributions, we generate a matrix of dimension $W_w \times H_w$, the height and width dimensions of the semantic map $\mathbf{S}^W_t$, and normalize the values to compute the distribution. To generate these matrices we use LINGUNET, a language-conditioned image-to-image encoder-decoder architecture [8].

The input to LINGUNET is the semantic map $\mathbf{S}^W_t$ and a grounding map $\mathbf{R}^W_t$ that incorporates the instruction u into the semantic map. We create $\mathbf{R}^W_t$ with a 1×1 convolution $\mathbf{R}^W_t = \mathbf{S}^W_t \circledast \mathbf{K}_G$. The kernel $\mathbf{K}_G$ is computed using a learned linear transformation $\mathbf{K}_G = \mathbf{W}_G \mathbf{u} + \mathbf{b}_G$, where $\mathbf{u}$ is the instruction embedding.

The grounding map $\mathbf{R}^W_t$ has the same height and width as $\mathbf{S}^W_t$, and during training we optimize the parameters so it captures the objects mentioned in the instruction u (Section 5).

LINGUNET uses a series of convolution and deconvolution operations. The input map $\mathbf{F}_0 = [\mathbf{S}^W_t, \mathbf{R}^W_t]$ is processed through L cascaded convolutional layers to generate a sequence of feature maps $\mathbf{F}_k = \text{CNN}_k(\mathbf{F}_{k-1})$, $k = 1 \dots L$.[1] Each $\mathbf{F}_k$ is filtered with a 1×1 convolution with weights $\mathbf{K}_k$. The kernels $\mathbf{K}_k$ are computed from the instruction embedding $\mathbf{u}$ using a learned linear transformation $\mathbf{K}_k = \mathbf{W}^u_k \mathbf{u} + \mathbf{b}^u_k$. This generates l language-conditioned feature maps $\mathbf{G}_k = \mathbf{F}_k \circledast \mathbf{K}_k$, $k = 1 \dots L$. A series of L deconvolution operations computes L feature maps of increasing size:

$$\mathbf{H}_k = \begin{cases} \text{DECONV}_k([\mathbf{H}_{k+1}, \mathbf{G}_k]), & \text{if } 1 \leq k \leq L-1 \\ \text{DECONV}_k(\mathbf{G}_k), & \text{if } k = L \end{cases} ,$$

The output of LINGUNET is $\mathbf{H}_1$, which is of size $W_w \times H_w \times 2$. The full details of LINGUNET are specified in Misra et al. [8]. We apply a softmax operation on each channel of $\mathbf{H}_1$ separately to generate the trajectory-visitation distribution d^p_t and the goal-visitation distribution d^g_t. In Section 5, we describe how we estimate the parameters to ensure that d^p and d^g model the visitation distributions.

4.2 Stage 2: Plan Execution

The action generation component $\text{ACT}(d^p_t, d^g_t, P_t)$ generates the action values from the two visitation distributions d^p_t and d^g_t and the current agent pose P_t. We first perform an affine transformation of the most recent visitation distributions to align them with the current agent egocentric reference frame as defined by its pose P_t, and crop a $K \times K$ region centered around the agent's position. We fill the positions outside the semantic map with zeros. We flatten and concatenate the cropped regions of the distributions into a single vector $\mathbf{x}$ of size $2K^2$, and compute the feed-forward network:

$$e^{\text{stop}}_t, v_t, \omega_t \quad = \quad \mathbf{W}^{(2)}[\mathbf{x}; \text{LEAKYRELU}\{0, \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\}] + \mathbf{b}^{(2)} \quad ,$$

where $[;]$ denotes concatenation of vectors and LEAKYRELU is a leaky ReLU non-linearity [47]. If the stopping probability $p^{\text{stop}}_t = \sigma(e^{\text{stop}}_t)$ is above a threshold κ the agent takes the stop action. Otherwise, we set the controller configuration using the forward velocity v_t and angular velocity ω_t.

5 Learning

Our model parameters can be divided into two groups. The visitation prediction $\text{VISIT}(\cdot)$ parameters θ_1 include the parameters of the functions ϕ, LSTM, and RESNET, $\mathbf{W}_G$, $\mathbf{b}_G$, and the components of LINGUNET: $\{\text{CNN}_k\}^L_{k=1}$, $\{\text{DECONV}_k\}^L_{k=1}$, $\{\mathbf{W}^u_k\}^L_{k=1}$, $\{\mathbf{b}^u_k\}^L_{k=1}$. The plan execution $\text{ACT}(\cdot)$ parameters θ_2 are $\mathbf{W}^{(1)}$, $\mathbf{b}^{(1)}$, $\mathbf{W}^{(2)}$, $\mathbf{b}^{(2)}$. We use supervised learning to estimate the visitation prediction parameters and imitation learning for the plan execution parameters.

[1] $[\cdot, \cdot]$ denotes concatenation along the channel dimension.

Estimating Visitation Prediction Parameters We assume access to training examples (u, s_1, Ξ), where u is an instruction, s_1 is a start state, and $\Xi = \langle p_1, \ldots, p_T \rangle$ is a sequence of T positions.[2] We convert the sequence Ξ to a sequence of positions in the semantic map $\tilde{\Xi} = \langle \tilde{p}_1, \ldots, \tilde{p}_T \rangle$. We generate expert trajectory-visitation distribution d^p_* by assigning high probability for positions around the demonstration trajectory, and goal-visitation distribution d^g_* by assigning high probability around the goal position p_T. For each location $\tilde{p} = (x, y)$ in the semantic map, we calculate the probability of visiting and stopping there as:

$$d^p_*(\tilde{p}) = \frac{1}{Z_p} \sum_{\tilde{p}_t \in \tilde{\Xi}} g(\tilde{p}|\tilde{p}_t, \sigma) \qquad d^g_*(\tilde{p}) = \frac{1}{Z_g} g(\tilde{p}|\tilde{p}_T, \sigma) \ ,$$

where $g(\cdot|\mu, \sigma)$ is a Gaussian probability density function with mean μ and variance σ^2, and Z_p and Z_g are normalization terms. The distributions are computed efficiently by applying a Gaussian filter on an image of the human trajectory. We then generate a sequence of agent contexts c_t by executing an oracle policy π^*, which is implemented with a simple control rule that steers the quadcopter along the human demonstration trajectory Ξ. We create a training example (c_t, d^p_*, d^g_*) for each time step $t = 1, T_d + 1, 2T_d + 1, \ldots$ in the oracle trajectory when we compute the visitation distributions, and minimize the KL divergence between the expert and predicted distribution: $D_{KL}(d^p_* \,||\, d^p(\cdot \mid c_t)) + D_{KL}(d^g_* \,||\, d^g(\cdot \mid c_t))$. The data and objective do not consider the incremental update of the distributions, and we always optimize towards the full visitation distributions.

We additionally use three auxiliary loss functions from Blukis et al. [7] to bias the different components in the model to specialize as intended: (a) the object recognition loss J_{percept} to classify visible objects using their corresponding positions in the semantic map; (b) the grounding loss J_{ground} to classify if a visible object in the semantic map is mentioned in the instruction u; and (c) the language loss J_{lang} to classify if objects are mentioned in the instruction u. To compute J_{ground} and J_{lang}, we use alignments between words and object labels that we heuristically extract from the training data using pointwise mutual information. Please refer to the supplementary material for full details.

The complete objective for an example (c_t, d^p_*, d^g_*) for time t is:

$$J(\theta_1) = D_{KL}(d^p_* \,||\, d^p(\cdot \mid c_t)) + D_{KL}(d^g_* \,||\, d^g(\cdot \mid c_t)) + \\ + \lambda_{\text{percept}} J_{\text{percept}}(\theta_1) + \lambda_{\text{ground}} J_{\text{ground}}(\theta_1) + \lambda_{\text{lang}} J_{\text{lang}}(\theta_1) \ ,$$

where $\lambda_{(\cdot)}$ is a hyperparameter weighting the contribution of the corresponding auxiliary loss.

Estimating Plan Execution Parameters We train the plan execution stage $\textsc{Act}(d^p, d^g, P)$ using imitation learning with the oracle policy π^*. During imitation learning, we use the visitation distributions d^p_* and d^g_* induced from the human demonstrations. This provides the model access to the same information that guides the oracle policy, which it learns to imitate. We use DAggerFM [7], a variant of DAgger [48] for low-memory usage. DAggerFM performs K iterations of training. For each iteration k and a training example (u, s_1, Ξ), we generate an execution $\langle (s_1, a_1), (s_2, a_2) \cdots (s_T, a_T) \rangle$ using a mixture policy. The mixture policy selects an action at time t using π^* with probability β^k or the learned policy $\textsc{Act}(d^p_*, d^g_*, P_t)$ with probability $1 - \beta^k$, where $\beta \in (0, 1)$ is a hyperparameter. The states generated in the execution are aggregated in a dataset across iterations. After each iteration, we prune the dataset to a fixed size and perform one epoch of supervised learning. We use a binary cross-entropy loss for the `STOP` probability p^{stop}, and a mean-squared-error loss for the velocities. When the oracle selects `STOP`, both velocities are zero. We initialize imitation learning with supervised learning using the oracle policy π^* trajectories.

Discussion Our approach is an instance of learning state-visitation distributions in Markov Decisions Processes (MDP). Consider an MDP $\langle \mathcal{S}, \mathcal{A}, R, \mathcal{T}, H, \mu \rangle$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $R : \mathcal{S} \to [0, R_{max}]$ is a reward function, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to Pr(\mathcal{S})$ is a probabilistic transition function, H is the time horizon, and μ is the start-state distribution.[3] The state-visitation distribution of a policy $\pi : \mathcal{S} \to Pr(\mathcal{A})$ is defined as $d(s; \pi, \mu) = \frac{1}{H} \sum_t d_t(s; \pi, \mu)$, where $d_t(s; \pi, \mu)$ is the probability of visiting state s at time t following policy π with the initial state-distribution μ.

Reasoning about the entire state space $\mathcal{S}$ is challenging. Instead, we consider an alternative discrete state space $\tilde{\mathcal{S}}$ with a mapping $\phi : \mathcal{S} \to \tilde{\mathcal{S}}$ and a reward function $\tilde{R} : \tilde{\mathcal{S}} \to \mathbb{R}^+$. For example, in a

[2] To simplify notation, we describe learning for a single example.

[3] $Pr(\cdot)$ denotes a probability distribution.

robot navigation scenario, $\tilde{s}$ can be the robot pose estimate $\tilde{s} = P$, or the positions in our semantic map $\mathbf{S}^W$. In a manipulation setup, $\tilde{s}$ can be the manipulator configuration. This choice is task-specific, but should include variables that are are measurable and relevant to task completion. The state-visitation distribution in $\tilde{\mathcal{S}}$ is $\tilde{d}(\tilde{s}; \pi, \mu) = \int \mathbb{1}\{\phi(s) = \tilde{s}\} d(s; \pi, \mu) ds$. In general, we construct $\tilde{S}$ as a small set to support efficient computation of the visitation distribution, and enable our two stage learning. In the first stage, we train a visitation model to predict the visitation distribution $\tilde{d}(\cdot; \pi^*, \mu)$ for the oracle policy π^*, and in the second stage, we learn a plan execution model using the oracle visitation distribution $\tilde{d}(\cdot; \pi^*, \mu)$ using imitation learning.

There is a strong relation between learning the state distribution and policy learning. For predicted visitation distributions $\hat{d}$ with a bounded error in regard to the optimal visitation distribution, the sub-optimality error of policies that accurately follow the predicted distribution is bounded as well:

Theorem 5.1. Suppose $D_{KL}(\tilde{d}(\cdot; \pi^*, \mu) \mid\mid \hat{d}) \leq \epsilon$ and let $\Pi(\eta) = \{\pi \mid D_{KL}(\hat{d} \mid\mid \tilde{d}(\cdot; \pi, \mu)) \leq \eta\}$ be the set of all policies whose approximate state-visitation distribution has at maximum η KL divergence from $\hat{d}$. Assume that for every $s \in \mathcal{S}$ there holds $|R(s) - \tilde{R}(\phi(s))| \leq \alpha$. Then:

$$\sup_{s \in \mathcal{S}} \sup_{\pi \in \Pi(\eta)} V^*(s) - V^\pi(s) \leq H\left(R_{max} + \alpha\right)\left(\sqrt{2\epsilon} + \sqrt{2\eta}\right) \quad .$$

6 Experimental Setup

Data and Environments We evaluate our approach on the LANI corpus [8]. LANI contains $27,965$ crowd-sourced instructions for navigation in an open environment. Each datapoint includes an instruction, a human-annotated ground-truth demonstration trajectory, and an environment with various landmarks and lakes. The dataset train/dev/test split is 19,758/4,135/4,072. Each environment specification defines placement of 6–13 landmarks within a square grass field of size 50m×50m. We use the quadcopter simulator environment from Blukis et al. [7] based on the Unreal Engine,[4] which uses the AirSim plugin [49] to simulate realistic quadcopter dynamics.

Data Augmentation We create additional data for visitation prediction learning by rotating the semantic map $\mathbf{S}^W$ and the gold distributions, d^p_* and d^g_*, by a random angle $\alpha \sim \mathcal{N}(0, 0.5\text{rad})$. This allows the agent to generalize beyond the common behavior of heading towards the object in front.

Evaluation Metric We measure the stopping distance of the agent from the goal as $\|p_g - p_T\|$, where p_g is the end-point of the human annotated demonstration and p_T is the position where the agent output the `STOP` action. A task is completed successfully if the stopping distance is $< 5.0m$, 10% of the environment edge-length. We also report the average and median stopping distance.

Systems We compare our Position-visitation Network (PVN) approach to the CHAPLOT [10] and GSMN [7] approaches. CHAPLOT is an instruction following model that makes use of gated attention. Similar to our approach, GSMN builds a semantic map, but uses simple language-derived convolutional filters to infer the goal location instead of computing visitation probabilities. We also report ORACLE performance as an upper bound and two trivial baselines: (a) STOP: stop immediately; and (b) AVERAGE: fly forward for the average number of steps (18) with the average velocity ($0.88m/s$), both computed with the ORACLE policy from the training data. Hyperparameter settings are provided in the supplementary material.

7 Results

Table 1 shows the performance on the test set and our ablations on the development set. The low performance of the STOP and AVERAGE baselines shows the hardness of the problem. Our full model PVN demonstrates absolute task-completion improvement of 16.85% over the second-best system (GSMN), and a relative improvement of 12.7% on average stopping distance and 32.3% on the median stopping distance. The relatively low performance of GSMN compared to previous results with the same environment but synthetic language [7], an accuracy drop of 54.8, illustrates the challenges introduced by natural language. The performance of CHAPLOT similarly degrades by 9.6 accuracy points compared to previously reported results on the same corpus but with a discrete environment [8]. This demonstrates the challenges introduced by a realistic simulation.

[4] https://www.unrealengine.com/

Method	SR (%)	AD	MD
Test Results			
STOP	5.72	15.8	14.8
AVERAGE	16.43	12.5	10.1
CHAPLOT	21.34	11.2	9.35
GSMN	24.36	9.94	8.28
PVN	**41.21**	**8.68**	**6.26**
ORACLE	100.0	1.38	1.29
Development Ablations and Analysis			
PVN	**40.44**	**8.56**	**6.28**
PVN NO AUX	30.77	10.1	7.94
PVN NO d^g	35.98	9.25	7.2
PVN NO DAGGER	38.87	9.18	6.69
PVN NO u	23.07	11.6	10.1
PVN IDEAL ACT	45.70	8.42	6.25
PVN FULL OBS	60.59	5.67	4.0
PVN height $\div$ 2	39.51	8.95	6.55
PVN $\omega \times 2$	41.09	8.6	6.12

Table 1: Test and development results, including model analysis. We evaluate success rate (SR), average stopping distance (AD), and median stopping distance (MD).

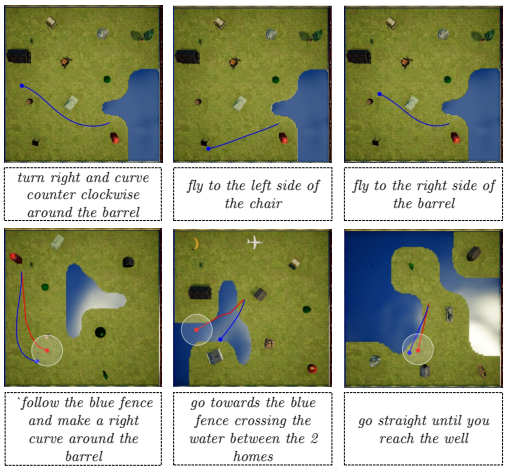

Figure 5: Our model executing engineered instructions from a single starting position (top) and representative instructions from the LANI development set (bottom). The maps show human demonstrations (red), our model trajectories (blue), and goal regions (white circles).

Our ablations show that all components of the methods contribute to its performance. Removing the auxiliary objectives (PVN NO AUX) or the goal-distribution prediction to rely only on the trajectory-visitation distribution (PVN NO d^g) both lower performance significantly. While using imitation learning shows a significant benefit, model performance degradation is less pronounced when only using supervised learning for the second stage (PVN NO DAGGER). The low performance of the model without access to the instruction (PVN NO u) illustrates that our model makes effective use of the input language. Figure 5 shows example trajectories executed by our model, illustrating the ability to reason about spatial language. The supplementary material includes more examples.

We evaluate the quality of goal-visitation distribution d^g with an ideal plan execution model that stops perfectly at the most likely predicted stopping position $\tilde{p}_g = \arg\max d^g(\tilde{p})$. The performance increase from using a perfect goal-visitation distribution with our model (PVN IDEAL ACT) illustrates the improvement that could be achieved by a better plan execution policy. We observe a more drastic improvement with full observability (PVN FULL OBS), where the input image I_t is set to the top-down view of the environment. This suggests the model architecture is capable of significantly higher performance with improved exploration and mapping.

Finally, we do initial tests for model robustness against test-time variations. We test for visual differences by flying at 2.5m (PVN height $\div$ 2), half the training height (5.0m). We test for dynamic differences by doubling the angular velocity during testing for every output action (PVN $\omega \times 2$). In both cases, the difference in model performance is relatively small, revealing the robustness of a modular approach to small visual and dynamics differences.

8 Conclusion

We study the problem of mapping natural language instructions and raw observations to continuous control of a quadcopter drone. Our approach is tailored for navigation. We design a model that enables interpretable visualization of the agent plans, and a learning method optimized for sample efficiency. Our modular approach is suitable for related tasks with different robotics agents. However, the effectiveness of our mapping mechanism with limited visibility, for example with a ground robot, remains to be tested empirically in future work. Investigating the generalization of our visitation prediction approach to other tasks also remains an important direction for future work.

Acknoledgements

This research was supported by Schmidt Sciences, NSF award CAREER-1750499, AFOSR award FA9550-17-1-0109, the Amazon Research Awards program, and cloud computing credits from Amazon. We thank the anonymous reviewers for their helpful comments.

References

[1] A. S. Huang, S. Tellex, A. Bachrach, T. Kollar, D. Roy, and N. Roy. Natural language command of an autonomous micro-air vehicle. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[2] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. Gopal Banerjee, S. Teller, and N. Roy. Approaching the Symbol Grounding Problem with Probabilistic Graphical Models. *AI Magazine*, 2011.

[3] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox. Learning to parse natural language commands to a robot control system. In *International Symposium on Experimental Robotics*, 2012.

[4] J. Thomason, S. Zhang, R. J. Mooney, and P. Stone. Learning to interpret natural language commands through human-robot dialog. In *International Joint Conferences on Artificial Intelligence*, 2015.

[5] D. Arumugam, S. Karamcheti, N. Gopalan, L. L. Wong, and S. Tellex. Accurately and efficiently interpreting human-robot instructions of varying granularities. In *Robotics: Science and Systems*, 2017.

[6] N. Gopalan, D. Arumugam, L. L. Wong, and S. Tellex. Sequence-to-sequence language grounding of non-markovian task specifications. In *Robotics: Science and Systems*, 2018.

[7] V. Blukis, N. Brukhim, A. Bennet, R. Knepper, and Y. Artzi. Following high-level navigation instructions on a simulated quadcopter with imitation learning. In *Robotics: Science and Systems*, 2018.

[8] D. Misra, A. Bennett, V. Blukis, E. Niklasson, M. Shatkin, and Y. Artzi. Mapping instructions to actions in 3D environments with visual goal prediction. In *Conference on Empirical Methods in Natural Language Processing*, 2018.

[9] D. Misra, J. Langford, and Y. Artzi. Mapping instructions and visual observations to actions with reinforcement learning. In *Conference on Empirical Methods in Natural Language Processing*, 2017.

[10] D. S. Chaplot, K. M. Sathyendra, R. K. Pasumarthi, D. Rajagopal, and R. Salakhutdinov. Gated-attention architectures for task-oriented language grounding. *AAAI Conference on Artificial Intelligence*, 2018.

[11] C. Matuszek, N. FitzGerald, L. Zettlemoyer, L. Bo, and D. Fox. A Joint Model of Language and Perception for Grounded Attribute Learning. In *International Conference on Machine Learning*, 2012.

[12] F. Duvallet, T. Kollar, and A. Stentz. Imitation learning for natural language direction following through unknown environments. In *IEEE International Conference on Robotics and Automation*, 2013.

[13] M. R. Walter, S. Hemachandra, B. Homberg, S. Tellex, and S. Teller. Learning Semantic Maps from Natural Language Descriptions. In *Robotics: Science and Systems*, 2013.

[14] D. K. Misra, J. Sung, K. Lee, and A. Saxena. Tell me dave: Context-sensitive grounding of natural language to mobile manipulation instructions. In *Robotics: Science and Systems*, 2014.

[15] 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 *IEEE International Conference on Robotics and Automation*, 2015.

[16] R. A. Knepper, S. Tellex, A. Li, N. Roy, and D. Rus. Recovering from Failure by Asking for Help. *Autonomous Robots*, 2015.

[17] M. MacMahon, B. Stankiewicz, and B. Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. In *AAAI Conference on Artificial Intelligence*, 2006.

[18] S. R. K. Branavan, L. S. Zettlemoyer, and R. Barzilay. Reading between the lines: Learning to map high-level instructions to commands. In *Annual Meeting of the Association for Computational Linguistics*, 2010.

[19] C. Matuszek, D. Fox, and K. Koscher. Following directions using statistical machine translation. In *International Conference on Human-Robot Interaction*, 2010.

[20] Y. Artzi and L. Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 2013.

[21] A. Suhr and Y. Artzi. Situated mapping of sequential instructions to actions with single-step reward observation. In *Annual Meeting of the Association for Computational Linguistics*, 2018.

[22] P. Shah, M. Fiser, A. Faust, J. C. Kew, and D. Hakkani-Tur. Followmet: Robot navigation by following natural language directions with deep reinforcement learning. *arXiv preprint arXiv:1805.06150*, 2018.

[23] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. v. d. Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. *arXiv preprint arXiv:1711.07280*, 2017.

[24] K. M. Hermann, F. Hill, S. Green, F. Wang, R. Faulkner, H. Soyer, D. Szepesvari, W. Czarnecki, M. Jaderberg, D. Teplyashin, et al. Grounded language learning in a simulated 3d world. *arXiv preprint*

arXiv:1706.06551, 2017.

[25] I. Lenz, H. Lee, and A. Saxena. Deep learning for detecting robotic grasps. *The International Journal of Robotics Research*, 2015.

[26] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen. Learning hand-eye coordination for robotic grasping with large-scale data collection. In *International Symposium on Experimental Robotics*, 2016.

[27] D. Quillen, E. Jang, O. Nachum, C. Finn, J. Ibarz, and S. Levine. Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods. *IEEE International Conference on Robotics and Automation*, 2018.

[28] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 2016.

[29] A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. In *IEEE International Conference on Robotics and Automation*, 2017.

[30] 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 *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[31] S. Bhatti, A. Desmaison, O. Miksik, N. Nardelli, N. Siddharth, and P. H. Torr. Playing doom with slam-augmented deep reinforcement learning. *arXiv preprint arXiv:1612.00380*, 2016.

[32] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn. Universal planning networks. *International Conference on Machine Learning*, 2018.

[33] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. *IEEE International Conference on Robotics and Automation*, 2018.

[34] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *Robotics: Science and Systems*, 2018.

[35] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[36] A. Khan, C. Zhang, N. Atanasov, K. Karydis, V. Kumar, and D. D. Lee. Memory augmented control networks. In *International Conference on Learning Representations*, 2018.

[37] N. Savinov, A. Dosovitskiy, and V. Koltun. Semi-parametric topological memory for navigation. *International Conference on Learning Representations*, 2018.

[38] 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.

[39] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *International Conference on Computer Vision*, 2017.

[40] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[41] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal. Learning and generalization of motor skills by learning from demonstration. In *IEEE International Conference on Robotics and Automation*, 2009.

[42] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal. Skill learning and task outcome prediction for manipulation. In *IEEE International Conference on Robotics and Automation*, 2011.

[43] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 2012.

[44] G. Maeda, M. Ewerton, T. Osa, B. Busch, and J. Peters. Active incremental learning of robot movement primitives. In *Conference on Robot Learning*, 2017.

[45] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann. Probabilistic movement primitives. In *Advances in Neural Information Processing Systems*. 2013.

[46] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[47] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, 2013.

[48] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International conference on artificial intelligence and statistics*, 2011.

[49] S. Shah, D. Dey, C. Lovett, and A. Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

A Details on Auxiliary Objectives

We use three additive auxiliary objectives to help the different components of the model specialize as intended with limited amount of training data.

Object Recognition Loss The object-recognition objective J_{percept} ensures the semantic map $\mathbf{S}_t^W$ stores information about locations and identities of various objects. At timestep t, for every object o that is visible in the first person image I_t, we classify the element in the semantic map $\mathbf{S}_t^W$ corresponding to the object location in the world. We apply a linear softmax classifier to every semantic map element that spatially corresponds to the center of an object. At a given timestep t the classifier loss is:

$$J_{\text{percept}}(\theta_1) = \frac{-1}{|O_{\text{FPV}}|} \sum_{o \in O_{\text{FPV}}} [\hat{y}_o log(y_o)] \ ,$$

where $\hat{y}_o$ is the true class label of the object o and y_o is the predicted probability. O_{FPV} is the set of objects visible in the image I_t.

Grounding Loss For every object o visible in the first-person image I_t, we use the feature vector from the grounding map $\mathbf{R}_t^W$ corresponding to the object location in the world with a linear softmax classifier to predict whether the object was mentioned in the instruction u. The objective is:

$$J_{\text{ground}}(\theta_1) = \frac{-1}{|O_{\text{FPV}}|} \sum_{o \in O_{\text{FPV}}} [\hat{y}_o log(y_o) + (1 - \hat{y}_o) log(1 - y_o)] \ ,$$

where $\hat{y}_o$ is a 0/1-valued label indicating whether the object o was mentioned in the instruction and y_o is the corresponding model prediction. O_{FPV} is the set of objects visible in the image I_t.

Language Loss The instruction-mention auxiliary objective uses a similar classifier to the grounding loss. Given the instruction embedding $\mathbf{u}$, we predict for each of the 63 possible objects whether it was mentioned in the instruction u. The objective is:

$$J_{\text{lang}}(\theta_1) = \frac{-1}{|O|} \sum_{o \in O_{\text{FPV}}} [\hat{y}_o log(y_o) + (1 - \hat{y}_o) log(1 - y_o)] \ ,$$

where $\hat{y}_o$ is a 0/1-valued label, same as above.

B Automatic Word-object Alignment Extraction

In order to infer whether an object o was mentioned in the instruction u, we use automatically extracted word-object alignments from the dataset. Let $E(o)$ be the event that an object o occurs within 15 meters of the human-demonstration trajectory Ξ, let $E(\tau)$ be the event that a word type τ occurs in the instruction u, and let $E(o, \tau)$ be the event that both $E(o)$ and $E(\tau)$ occur simultaneously. The pointwise mutual information between events $E(o)$ and $E(\tau)$ over the training set is:

$$\text{PMI}(o, \tau) = P(E(o, \tau)) \log \frac{P(E(o, \tau))}{P(E(o))P(E(\tau))} \ ,$$

where the probabilities are estimated from counts over training examples $\{(u^{(i)}, s_1^{(i)}, \Xi^{(i)})\}_{i=1}^N$. The output set of word-object alignments is:

$$\{(o, \tau) \mid \text{PMI}(o, \tau) > T_{\text{PMI}} \wedge P(\tau) < T_\tau\} \ ,$$

where $T_{PMI} = 0.008$ and $T_\tau = 0.1$ are threshold hyperparameters.

C Hyperparameter Settings

Image and Feature Dimensions
Camera horizontal FOV: 90°
Input image dimensions: $128 \times 72 \times 3$
Feature map $\mathbf{F}^C$ dimensions: $32 \times 18 \times 32$
Semantic map $\mathbf{S}^W$ dimensions: $64 \times 64 \times 32$
Visitation distributions d^g and d^p dimensions: $64 \times 64 \times 1$
Cropped visitation distribution dimensions: $12 \times 12 \times 1$

Environment edge length in meters: $50m$
Environment edge length in pixels on $\mathbf{S}^W$: 32

Model
Visitation prediction interval timesteps: $T_d = 6$
`STOP` action threshold: $\kappa = 0.07$

General Learning
Auxiliary objective weights: $\lambda_{\text{percept}} = 1.0$, $\lambda_{\text{ground}} = 1.0$, $\lambda_{\text{lang}} = 0.25$

Supervised Learning
Learning library: PyTorch 0.3.0
Optimizer: ADAM
Learning Rate: 0.001
Weight Decay: 10^{-6}
Batch Size: 1

Imitation Learning
Mixture decay: $\beta = 0.92$
Number of iterations: 100
Number of environments for policy execution per iteration: 10
Number of policy executions per iteration (executions): 47 on average
Memory size (number of executions): 600

D Proof of Theorem 5.1

Proof. Given that the state-visitation distribution of a policy $\pi : \mathcal{S} \rightarrow Pr(\mathcal{A})$ is defined as $d(s;\pi,\mu) = \frac{1}{H}\sum_t d_t(s;\pi,\mu)$, we can write the state-value function for the policy π as:

$$V^\pi(s) = H \int d(s';\pi,\delta_s)R(s')ds' \ ,$$

where δ_s is the start-state distribution that places the entire probability mass on state s.

Using the definition $V^\pi(s)$ and assuming $\pi \in \Pi(\eta)$ we can write,

$$\begin{aligned}
V^*(s) - V^\pi(s) &= H \int d(s';\pi^*,\delta_s)R(s')ds' - H \int d(s';\pi,\delta_s)R(s')ds' \\
&= H \int \{d(s';\pi^*,\delta_s) - d(s';\pi,\delta_s)\} R(s')ds' \\
&\leq H \int \{d(s';\pi^*,\delta_s) - d(s';\pi,\delta_s)\} (\tilde{R}(\phi(s')) + \alpha)ds' \\
&= H \int \{d(s';\pi^*,\delta_s) - d(s';\pi,\delta_s)\} \tilde{R}(\phi(s'))ds' + \\
&\quad H\alpha \int \{d(s';\pi^*,\delta_s) - d(s';\pi,\delta_s)\} ds' \\
&= H \int \{d(s';\pi^*,\delta_s) - d(s';\pi,\delta_s)\} R(\phi(s'))ds' \\
&\quad \text{Because } d \text{ is a probability distribution, which gives} \\
&\qquad \int \{d(s';\pi^*,\delta_s) - d(s';\pi,\delta_s)\}ds' = \\
&\qquad\qquad \int d(s';\pi^*,\delta_s)ds' - \int d(s';\pi,\delta_s)ds' = 0 \ . \\
&= H \sum_{\tilde{s}\in\tilde{\mathcal{S}}} \tilde{R}(\tilde{s}) \int \mathbb{1}\{\phi(s') = \tilde{s}\} \{d(s';\pi^*,\delta_s) - d(s';\pi,\delta_s)\} ds' \\
&= H \sum_{\tilde{s}\in\tilde{\mathcal{S}}} \tilde{R}(\tilde{s}) \left\{\tilde{d}(\tilde{s};\pi^*,\delta_s) - \tilde{d}(\tilde{s};\pi,\delta_s)\right\}
\end{aligned}$$

$$
\begin{aligned}
&\leq H \sum_{\tilde{s}\in\tilde{\mathcal{S}}} \left|\tilde{R}(\tilde{s})\left\{\tilde{d}(\tilde{s};\pi^*,\delta_s) - \tilde{d}(\tilde{s};\pi,\delta_s)\right\}\right| \\
&\leq H \left\{\sum_{\tilde{s}\in\tilde{\mathcal{S}}} \left|\tilde{d}(\tilde{s};\pi^*,\delta_s) - \tilde{d}(\tilde{s};\pi,\delta_s)\right|\right\} \max_{\tilde{s}\in\mathcal{S}} |\tilde{R}(\tilde{s})| \\
&\quad \text{Using Holder's inequality} \\
&= H \left\{\sum_{\tilde{s}\in\tilde{\mathcal{S}}} \left|\tilde{d}(\tilde{s};\pi^*,\delta_s) - \hat{d}(\tilde{s}) + \hat{d}(\tilde{s}) - \tilde{d}(\tilde{s};\pi,\delta_s)\right|\right\} \max_{\tilde{s}\in\mathcal{S}} |R(\tilde{s})| \\
&\leq H \left\{\sum_{\tilde{s}\in\tilde{\mathcal{S}}} \left|\tilde{d}(\tilde{s};\pi^*,\delta_s) - \hat{d}(\tilde{s})\right| + \sum_{\tilde{s}\in\tilde{\mathcal{S}}} \left|\hat{d}(\tilde{s}) - \tilde{d}(\tilde{s};\pi,\delta_s)\right|\right\} \max_{\tilde{s}\in\mathcal{S}} |R(\tilde{s})| \\
&\leq H \left(\sqrt{2D_{KL}(\tilde{d}(\cdot;\pi^*,\delta_s) \mid\mid \hat{d})} + \sqrt{2D_{KL}(\hat{d} \mid\mid \tilde{d}(\cdot;\pi,\delta_s))}\right) \max_{\tilde{s}\in\tilde{\mathcal{S}}} |\tilde{R}(\tilde{s})| \\
&\quad \text{Using Pinsker's inequality.} \\
&\leq H \left(\sqrt{2\epsilon} + \sqrt{2\eta}\right) \max_{\tilde{s}\in\tilde{\mathcal{S}}} |\tilde{R}(\tilde{s})| \\
&\quad \text{Using the theorem assumptions.} \\
&\leq H(R_{max} + \alpha)\left(\sqrt{2\epsilon} + \sqrt{2\eta}\right) \\
&\quad \text{Without loss of generality we assume } \tilde{R}(\tilde{s}) = 0 \text{ for } \tilde{s} \text{ s.t. there exists no } s, \\
&\quad \text{where } \phi(s) = \tilde{s}. \text{ Additionally, } \tilde{R} : \tilde{\mathcal{S}} \to \mathbb{R}^+ \text{ rewards are only positive.} \\
&\quad \text{Therefore, } \tilde{R}(\phi(s)) \leq R(s) + \alpha \Rightarrow \max_{\tilde{s}\in\tilde{\mathcal{S}}} |\tilde{R}(\tilde{s})| = \max_{\tilde{s}\in\tilde{\mathcal{S}}} \tilde{R}(\tilde{s}) \leq R_{max} + \alpha \quad .
\end{aligned}
$$

We did not use any information about π or s in the above steps except for $\pi \in \Pi(\eta)$. Therefore taking supremum over s and $\pi \in \Pi(\eta)$ completes the proof. □

E Additional instruction-following examples

Figure 6 shows example instructions from the development set along with the trajectories taken by our model and the human demonstrators.

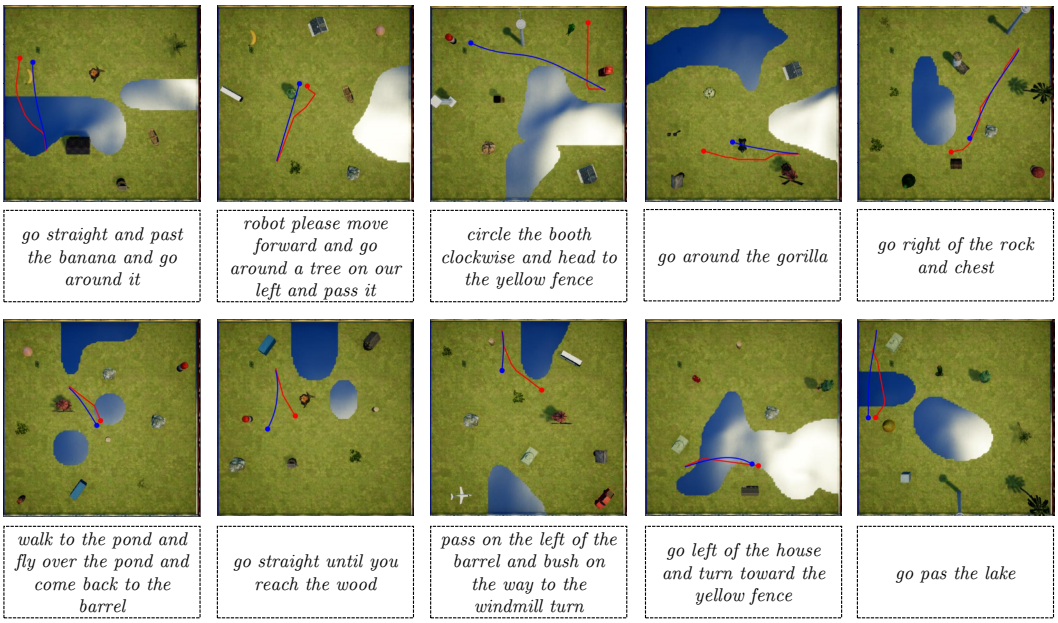

Figure 6: Instruction following results (blue) and human demonstration trajectories (red) on randomly selected instructions from the LANI development set.