Robotics: Science and Systems 2022
New York City, NY, USA, June 27-July 1, 2022

Correcting Robot Plans with Natural Language Feedback

Pratyusha Sharma[‡§], Balakumar Sundaralingam[‡], Valts Blukis[‡], Chris Paxton[‡],
Tucker Hermans[‡¶], Antonio Torralba[§], Jacob Andreas [§], Dieter Fox[‡†]
[‡] NVIDIA, [§] MIT, [¶] University of Utah, [†] University of Washington

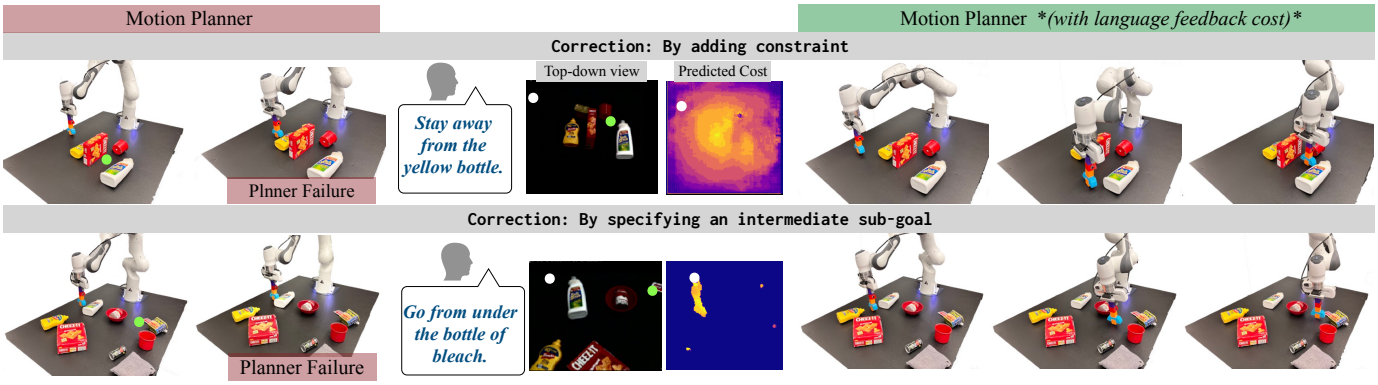


Fig. 1: Robots often fail to do what we want. This can happen for many reasons including mis-specification of goals, failure to anticipate what satisfying plans will do, and because optimization sometimes fails. We show how language can be used to update the underlying cost of a planner to improve task performance. Our approach can use language to specify corrections by a) the addition of constraints or b) specifying intermediate sub-goals for the planner .

***Abstract*—When humans design cost or goal specifications for robots, they often produce specifications that are ambiguous, under-specified, or beyond planners' ability to solve. In these cases, *corrections* provide a valuable tool for human-in-the-loop robot control. Corrections might take the form of new goal specifications, new constraints (e.g. to avoid specific objects), or hints for planning algorithms (e.g. to visit specific waypoints). Existing correction methods (e.g. using a joystick or direct manipulation of an end effector) require full teleoperation or real-time interaction. In this paper, we explore *natural language* as an expressive and flexible tool for robot correction. We describe how to map from natural language sentences to *transformations of cost functions*. We show that these transformations enable users to correct goals, update robot motions to accommodate additional user preferences, and recover from planning errors. These corrections can be leveraged to get 81% and 93% success rates on tasks where the original planner failed, with either one or two language corrections. Our method makes it possible to compose multiple constraints and generalizes to unseen scenes, objects, and sentences in simulated and real-world environments. Additional visualizations are available at sites.google.com/view/language-costs**

I. Introduction

Consider a robot vacuum cleaner. The robot's goal is to clean the house, but there may be a need to alter the objective (*"Clean only the living room."*), to introduce constraints (*"Don't go into the bathrooms!"*) or to guide the robot when it is stuck (*"Go to the right end of the wall to enter the missed room."*). The robot would benefit from the ability to incorporate such corrective, natural language feedback to alter aspects of its behavior or modify its goal. How though can the robot incorporate instructions with rich and varied semantics into its existing objective?

In this paper, we propose to use natural language instructions as inputs to directly modify a robot's planning objective. This objective function takes the form of a cost function in an optimization-based planning and control framework for manipulation. Our use of language contrasts with previous work where corrective input of robot behavior came from joystick control [36, 33], kinesthetic feedback [27, 19, 6], or spatial labelling of constraints [45, 9]. Kinesthetic and joystick feedback allows for fine-grained control, but typically requires prior expertise and undivided attention from the user, reducing the system autonomy and limiting its applications.

We choose natural language input thanks to its efficiency, accessibility, ease-of-use, and direct conveyance of the user's intent [39]. This allows for the expression of a broad range of feedback describing physical variation, time, or other abstractions. However, language also brings with it ambiguity and requires a model of symbol grounding and spatio-temporal reasoning to relate the concepts expressed in language to the robot's state and action spaces [17, 25, 37, 22, 3, 30, 29]. Given such a grounding model, the robot can ground language corrections to novel tasks and environments, achieving powerful generalization. In contrast, learning to generalize corrections from kinesthetic or joystick data to novel tasks requires inferring the user's intent given few underspecified demonstrations, which is itself a challenging problem [14].

We propose to learn a model that maps visual observations and a natural language correction to a residual cost function. Specifically, we model natural-language corrections as a residual cost function that can be combined with a task cost. This allows a user to modify the robot's objective, to clarify a misspecified objective, or introduce additional constraints into the motion optimization process at varying levels of abstraction at any time during execution. In the absence of a prior task objective, our method can also be used to specify the task in an instruction-following setting [23, 41]. Our

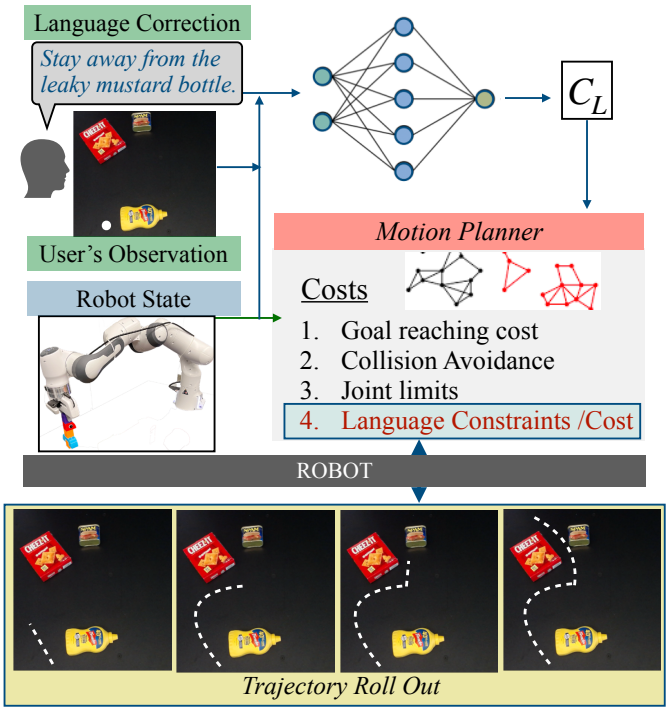


Fig. 2: Setup: The main components of our system are: the motion planner and language parameterized cost correction module. The motion planner uses sampling based model predictive control to minimize the overall specified cost. The cost correction module takes as input its observation of the environment, the robot's state, and a natural language correction to output an updated cost function. The motion planner uses the updated cost to modify the trajectory of the robot.

framework seamlessly integrates with commonly used motion planner costs, such as collision avoidance, joint limits, and smoothness. It also allows layering costs sequentially or at a given time, allowing for time-varying corrections. Finally, it enables composing costs associated with previously learned tasks or corrections to represent new tasks at a higher level of abstraction. We train our cost model on a dataset of natural language descriptions paired with either demonstrations, pre-specified costs, or both.

We conduct experiments both in simulation and on the physical robot manipulator illustrated in Figure 1. These experiments show how we can use our method to either specify or correct robot behavior with natural language commands. In environments where our local planner fails 6.4% of the time, humans can use our language interface to correct 81% of failures with one natural language command and 93% with a second, bringing the effective success rate from 94% to 99%. Our method generalizes to unseen objects and to out-of-distribution natural language, and the cost maps it creates are composable, meaning that commands can be combined with various cost functions. Finally, we show how our method, trained in simulation, can be applied to real-robot tasks.

II. Preliminaries

Given an environment $\mathcal{E}$, we study planning problems formalized as Markov decision processes defined by the tuple $(\mathcal{S}, \mathcal{A}, \Omega, T)$. Where $\mathcal{S}$ is the set of robot states, $\mathcal{A}$ is the set of possible actions, Ω is the space of external observations, and $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the stochastic transition function. In the case of the robot in Figure 2, $\mathcal{S}$ is a tuple of all possible positions, velocities and accelerations the robot can achieve represented as $(q, \dot{q}, \ddot{q})$. The action space $\mathcal{A} = \{up, down, left, right\}$. Its Ω is the observed cost of states k-steps around the robot's current s. The tuple $(s, a, o, \mathcal{T})$ describes the robot's state $s \in \mathcal{S}$, action taken $a \in \mathcal{A}$, observations from the environment $o \in \Omega$ and the task $\mathcal{T}$ that the robot needs to complete. For a $\mathcal{T}$, an associated cost map represented as $\mathcal{C}_\mathcal{T}$ may be specified by a user that wants a planner to perform $\mathcal{T}$. $\mathcal{C}_\mathcal{T}$ is of the form,

$$\mathcal{C}_\mathcal{T} : \mathcal{S} \rightarrow \mathbb{R}$$

and is a user specified costmap which can be probed for different $s \in \mathcal{S}$. For example, for reaching a goal, g in Figure 1 the costmap is a function that at every state on the map s returns the Euclidean distance $\sqrt{(s-g)^2}$. In addition to the $\mathcal{C}_\mathcal{T}$, there is also a base cost $\mathcal{C}_\mathcal{B}$[1] that helps the robot avoid its limits (collision avoidance, joint limits). Conditioned on o the robot takes an a sequentially to minimize,

$$\mathcal{C}_\mathcal{R} = \mathcal{C}_\mathcal{T} + \mathcal{C}_\mathcal{B}. \tag{1}$$

The planning routine $\mathcal{P}$ then optimizes the robot's cumulative cost $\mathcal{C}_\mathcal{R}$ and outputs an estimate of the best action that can be taken in the given state s. The robot then takes the action a and ends up in a new state dictated by the dynamics of the robot T and the system and the processes is repeated. It is via this process that robot unrolls a trajectory, τ, to complete different tasks. More formally, $\mathcal{P}$ is,

$$\begin{aligned} \underset{\ddot{q}_t \in [0,T]}{\arg\min} \quad & \sum_{t=0}^{T} \mathcal{C}_\mathcal{T}(q_t, \dot{q}_t) + \mathcal{C}_\mathcal{B}(q_t, \dot{q}_t) \\ \text{s.t.} \quad & q_t = q_{t-1} + \dot{q}_t dt \qquad (2) \\ & \dot{q}_t = \dot{q}_{t-1} + \ddot{q}_t dt \qquad (3) \end{aligned}$$

With increasing complexity of the environment and without assuming access to a model of the world, it is challenging to specify a $\mathcal{C}_\mathcal{T}$ that accurately reflects the task. An optimal cost function would be one that reflects the intended task, capturing the true cost-to-go. For example, the cost function corresponding to navigating to a goal could be approximated as the euclidean distance to goal. However, in many environments, as shown in Fig 1, greedily optimizing this objective will result in only a locally optimal solution that is completely misaligned with the human's intent. Misalignment due to misspecified goals and insufficient constraints can cause a plan failure. This problem of misalignment or a difference in the cost in the mind of the human $\mathcal{C}_\mathcal{H}$ and the robot $\mathcal{C}_\mathcal{R}$ is referred

[1]Detailed specifications of $\mathcal{C}_\mathcal{B}$ can be found in the appendix A

to as the value alignment problem [43]. Feedback from the human can be used to minimize the differences between $\mathcal{C}_\mathcal{H}$ and $\mathcal{C}_\mathcal{R}$. The human can generate a feedback based on their observation of the $\mathcal{E}$, represented as o^h and other variables of the $\mathcal{E}$ they have access to.

III. Approach

To minimize the gap between $\mathcal{C}_\mathcal{H}$ and $\mathcal{C}_\mathcal{R}$, we propose the use of feedback from a user in the form of *natural language corrections* to update $\mathcal{C}_\mathcal{R}$. Below we outline our approach.

A. Our Method

At any given point of time t, the user can issue feedback in the form of a natural language string, denoted as $\mathcal{L}$. We assume that the user has access to o^h, s, and information about the task while generating feedback. We learn a generative model that generates a costmap over all $states$ associated with $\mathcal{L}$ conditioned on $\mathcal{L}$, s the state of the robot, and o^h. This cost is then composed with $\mathcal{C}_\mathcal{R}$ ($\mathcal{C}^*_\mathcal{R} = \mathcal{C}_\mathcal{R} + \mathcal{C}_\mathcal{L}$) to generate an updated cost for the robot. $\mathcal{P}$ then solves the optimization informed by the updated objective $\mathcal{C}^*_\mathcal{R}$.

We factor the language-based cost $\mathcal{C}_\mathcal{L}$ into functions that generate a continuous cost map $\mathcal{C}$ and a binary mask over the cost $\mathcal{M}$. We combine them using element-wise multiplication, $\mathcal{C}_\mathcal{L} = \mathcal{C} * \mathcal{M}$. Where the functions have the form,

$$\mathcal{C} : \mathcal{S} \rightarrow \mathbb{R}$$
$$\mathcal{M} : \mathcal{S} \rightarrow [0, 1].$$

$\mathcal{C}$ for a given $\mathcal{L}$ maps to the cost for every $s \in \mathcal{S}$. $\mathcal{M}$ maps to a binary mask that is used over $\mathcal{C}$. In the case of a goal-directed $\mathcal{L}$, such as *go to the left of the bottle*, this is a guiding path to the goal. Whereas in the case of a constraints such as, *go slower*, this is a unit mask i.e. no-states are masked. This is done in-order to use a cue from the mask to help distinguish between changes in goals versus adding constraints to existing goals better. In theory, the cost-map itself should be able to direct a robot to the goal but we see that learning such a decomposition worked better in practice, specially in long-horizon tasks as shown in Appendix C.

$\mathcal{C}$ and $\mathcal{M}$ are learnt using datasets containing data of one or both types. Dataset containing trajectories paired with $\mathcal{L}$, $\mathcal{D}_{\text{demos}} = \{(\tau_1, \mathcal{L}_1, o^h_1, s_1), ...(\tau_n, \mathcal{L}_n, o^h_n, s_n)\}$, and costmaps paired with $\mathcal{L}$, $\mathcal{D}_{\text{cmap}} = \{(\mathcal{C}_1, \mathcal{L}_1, o^h_1, s_1), ...(\mathcal{C}_k, \mathcal{L}_k, o^h_k, s_k)\}$. Using $\mathcal{D}_{demos}$ and $\mathcal{D}_{cmap}$ we generate a unified final dataset.

1) Generating Ground-truth $\mathcal{C}$ and $\mathcal{M}$: The $\mathcal{C}$ associated with trajectories τ in $\mathcal{D}_{\text{demos}}$ is generated by mapping every s on the trajectory to its distance to the goal measured along the trajectory and every s outside the trajectory is mapped to a fixed high cost. This kind of a mapping is representative of the fact that moving along the trajectory is definitively indicative of a decrease in cost for the specified $\mathcal{L}$. For tasks where cost maps are specified, for instance *stay away from X*, the cost maps available are used directly. The process of generating ground truth masks for training is fairly straight forward. For datapoints in $\mathcal{D}_{\text{demos}}$, the binary mask is 1 for states along τ and is zero everywhere else. For datapoints in $\mathcal{D}_{\text{cmap}}$ the mask is I. The final dataset is $\mathcal{D} = \{(\mathcal{C}_1, \mathcal{M}_1, \mathcal{L}_1, o^h_1, s_1), ...\}$ where c and m correspond to the cost map and binary mask corresponding to the datapoint.

2) Objective: $\mathcal{C}$ and $\mathcal{M}$ are learnt on the dataset $\mathcal{D}$ via maximum likelihood estimation. We initialize the parameters for models that learn $\mathcal{C}$ and $\mathcal{M}$ with parameters θ and η. The models for $\mathcal{C}$ and $\mathcal{M}$ condition on $\mathcal{L}$, s, and o^h. The probability of generating the correct $\mathcal{C}$ and $\mathcal{M}$ can be decomposed as follows.

$$p(\mathcal{C} \mid \mathcal{L}_i, s_i, o^h_i) = \prod_s p\Big(\mathcal{C}(s) \mid \mathcal{L}_i, s_i, o^h_i\Big) \tag{4}$$

$$p(\mathcal{M} \mid \mathcal{L}_i, s_i, o^h_i) = \prod_s p\Big(\mathcal{M}(s) \mid \mathcal{L}_i, s_i, o^h_i\Big) \tag{5}$$

To update model parameters the optimization objective is,

$$\theta = \arg\max_\theta \sum_{i=1}^{n+k} \sum_{s \in \mathcal{S}} \log p_\theta\Big(\mathcal{C}_i(s) \mid ((\mathcal{L}_i, s_i, o^h_i), s)\Big) \tag{6}$$

$$\eta = \arg\max_\eta \sum_{i=1}^{n+k} \sum_{s \in \mathcal{S}} \log p_\eta\Big(\mathcal{M}_i(s) \mid ((\mathcal{L}_i, s_i, o^h_i), s)\Big) \tag{7}$$

For datapoints from $\mathcal{D}_{\text{demos}}$ we only penalize the cost prediction model for s on the trajectory whereas for datapoints from $\mathcal{D}_{\text{cmap}}$ we penalize the model for the entire costmap. This is because, for demonstrations we are only confident about costs along the trajectory. This partial supervision used while training allows the model to extrapolate and make guesses of the costs everywhere else in the map and as a result the cost maps generated are smoother. At inference, to obtain the $\mathcal{C}$ and $\mathcal{M}$ corresponding to a new $\mathcal{L}$, o^h and s,

$$\mathcal{C} = \arg\max_\mathcal{C} \; p_\theta(\mathcal{C} \mid \mathcal{L}_i, s_i, o^h_i) \tag{8}$$

$$\mathcal{M} = \arg\max_\mathcal{M} \; p_\eta(\mathcal{M} \mid \mathcal{L}_i, s_i, o^h_i) \tag{9}$$

3) Interfacing $\mathcal{C}_\mathcal{L}$ with the $\mathcal{P}$: We explore two corrections types that can be encoded by our $\mathcal{C}_\mathcal{L}$; constraint addition and goal specification. In the case of constraint addition, the constraints are added to the $\mathcal{P}$ permanently (e.g., going faster, slower, staying away from an object). While optimizing, we keep track of the constraints in a set $\mathcal{C}_{\mathcal{LC}}$ to enable accounting over multiple constraints. In the case of goal specification, there are two cases in which a goal may be specified, first, in the absence of any previous goal and second, as a way to correct the model by introducing intermediate goals. In the first case, there is no existing goal cost and $\mathcal{C}_\mathcal{L}$ becomes the only active goal cost alongside the other constraints. In the second mode we deactivate the task cost $\mathcal{C}_\mathcal{T}$ and wait until $\mathcal{C}_\mathcal{L}$ is within a threshold ϵ before activating the original task cost $\mathcal{C}_\mathcal{T}$ back again. This temporary activation mode is used where $\mathcal{L}$ specifies an intermediate goal directive. The $\mathcal{M}$ along with the presence or absence of an existing $\mathcal{C}_\mathcal{T}$ is indicative of the mode of correction. A $\mathcal{L}$ with a $\mathcal{M} == 1$ is always a constraint and that with a $\mathcal{M} \neq 1$ is a goal directive. We interface our $\mathcal{C}_\mathcal{L}$ with an optimization based controller [7] to generate commands for the robot as shown in Algorithm 1.

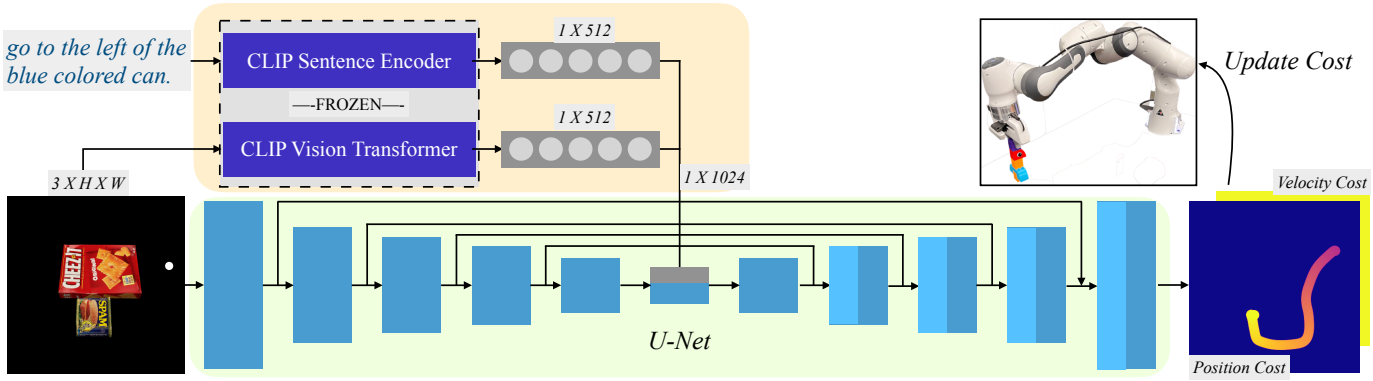


Fig. 3: Architecture: The architecture of the language parametrized cost correction module consists of two streams . The CLIP stream takes as input the natural language feedback as well as an image of the environment and the U-Net stream encodes the image of the environment. The output of this model is used to map to the cost associated with the language instruction. This could be learnt using specified costs corresponding to specific instructions or via estimating the cost from demonstrations. More details in section section III

Algorithm 1: $\mathcal{C}_{\mathcal{L}}$ interfaced with the $\mathcal{P}$

Initialize $\mathcal{C}_{\mathcal{LC}} = []$, $\mathcal{C}_{\mathcal{L}} = 0$, $\mathcal{C}^*_{\mathcal{R}} = \mathcal{C}_{\mathcal{T}} + \mathcal{C}_{\mathcal{B}} + \mathcal{C}_{\mathcal{LC}}$
while *task not done* **do**
  s_t = get_new_state() // Get current robot state;
  if *user feedback* **then**
    $\mathcal{L}$ = get_user_feedback();
    $\mathcal{C} = p_\theta\ (\mathcal{L}, s_t, o^h_t)$
    $\mathcal{M} = p_\eta\ (\mathcal{L}, s_t, o^h_t)$ // Run inference; **if** $\mathcal{M} == I$ **then**
      $\mathcal{C}_{\mathcal{LC}}$.append($\mathcal{C} * \mathcal{M}$) // Add to constraints list;
    else
      $\mathcal{C}_{\mathcal{L}} = \mathcal{C} * \mathcal{M}$
      $\mathcal{C}^*_{\mathcal{R}} = \mathcal{C}_{\mathcal{T}} + \mathcal{C}_{\mathcal{B}} + \mathcal{C}_{\mathcal{L}} + \mathcal{C}_{\mathcal{LC}}$
    end
    if $\mathcal{C}_{\mathcal{L}}(s_t) < \epsilon$ **then**
      $\mathcal{C}^*_{\mathcal{R}} = \mathcal{C}_{\mathcal{T}} + \mathcal{C}_{\mathcal{B}} + \mathcal{C}_{\mathcal{LC}}$ // Original goal-cost is active;
  $a_{t+1} = \mathcal{P}(\mathcal{C}^*_{\mathcal{R}}, s_t)$ // Optimize with planner;
  command_robot(a_{t+1}) // Send command to robot;
  t+=1
end

B. Architecture

$\mathcal{C}$ and $\mathcal{M}$ are both individually implemented as a neural network with a two-stream architecture as seen in Fig 3. The CLIP stream consists of a pre-trained Contrastive Language-Image Pre-training (CLIP) [32] model with the Vision Transformer(ViT) [13] visual module. It takes as input the language correction $\mathcal{L}$, robot state s and the RGB representation o^h. The state of the robot is encoded using the location of it's end effector on a spatial map of the RGB o^h. We use the 512 dimensional visual embedding output from ViT along with the 512 dimension language embedding output from the language-transformer. The image is encoded using a U-Net architecture with skip connections [34]. It encodes the RGB image o^h and robot state s of the robot and generates the output frames corresponding to the position cost map and velocity cost map each parametrized as 2D map in $\mathcal{R}^{|\mathcal{S}|2}$. The visual and language embedding from CLIP are concatenated with the embedding of the encoder before passing the embedding through the deconvolution layers to generate the cost map. The weights of the CLIP language transformer and ViT are frozen and the training optimizes the weights of the U-Net only. The model outputs two cost maps: 1) a position map and 2) a velocity map.[3]

IV. Experimental Protocol

For $\mathcal{D}_{\text{demos}}$, we generate a dataset containing 100 environments. Each environment contained two objects from a set of four YCB objects [11]: a Cheeze-it box, a bleach bottle, a can of spam and a bottle of mustard. The position of each object is sampled uniformly within the bounds of the environment. We sample object orientation from one of four equally-spaced options. We render each environment with the NVIDIA Scene Imaging Interface (NViSII) [26]. Images are top-down and are of size $(2048, 2048)$ pixels. For every environment, we uniformly sample 10 different collision-free start positions. We choose goal positions to be 20 pixels offset from the mid-point of object edges where the offset is away from the object, and generate corresponding templated language instructions. This templated language is sampled from diverse referring expressions and object descriptions, as shown in Fig. 4.

We use STORM [7] as a planner which minimizes a Euclidean distance cost to generate trajectories from start to goal position. However, as this is not a global optimizer, it can get caught in local minima. Trajectories that successfully connect these positions are categorized as successful; failed trajectories are stored separately as a *hard* set for evaluation. In our setting, the planner failed 6.4% of the time. More details on the planner is described in appendix A. We then divided the environments with successful trajectories into training, validation, and test sets, so that a specific object configuration will only appear in one split.

[2] $|\mathcal{S}|$ denotes the dimension of the $\mathcal{S}$

[3] Our framework can also be extended to output additional maps for new quantities such as force and torque by adding additional frames to the output.

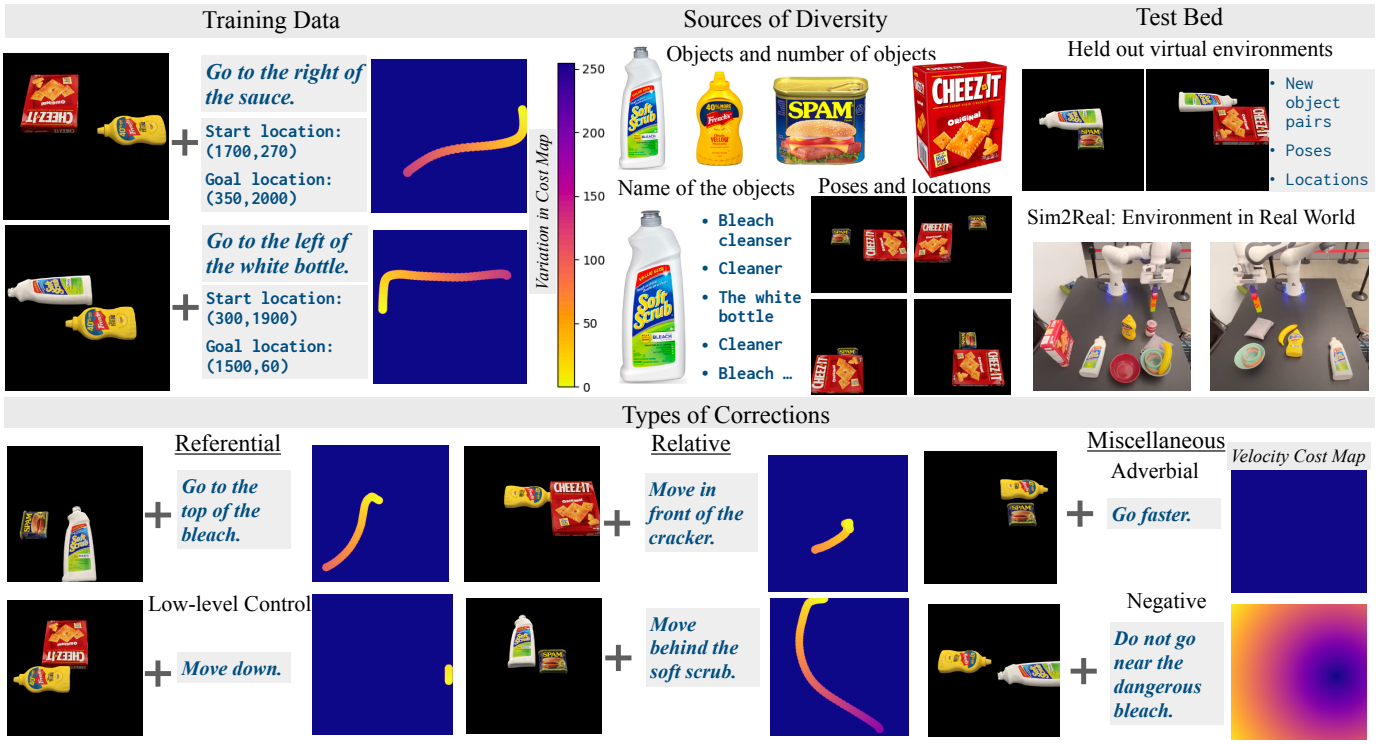


Fig. 4: Dataset: Clockwise starting from top left a) The training data consists of data of the form (input:(instructions, environments,robot state), output:(cost)). The cost can be either specified by the user directly or come from demonstrations. When learnt from the demonstration, the trajectory is modified by linearly decreasing the cost from the start location to the end location along the trajectory. b) The sources of diversity in the dataset comes from the types of objects, randomized positioning of the objects in the environment and from instructions templated to refer to the same object in a variety of different ways as shown c) There are two testing setups. Unseen environments with new object combinations, object locations in simulation and a real environment setup with a 7DoF robot arm. d) The vocabulary of corrections include referential expressions, relative clauses and miscellaneous interesting correction types such as avoid the object or costs that modify non-positional attributes of the robot's behavior such a velocity.

For $\mathcal{D}_{\text{cmap}}$, the cost maps are generated in the following way. For velocity speed-up, slow down, and when no constraints are given over velocity the velocity costmap outputs 0, 1, or 2 respectively, corresponding to all $s \in \mathcal{S}$. For instructions of avoiding objects the cost map generated is $-\sqrt{(s-c)^2}$ where c is the location of the centre of the object to be stayed away from. All cost values are re-scaled between $[0, 255]$.

Evaluation and metrics. We consider a trial a success when the robot reaches within 20 pixels of the goal position. We evaluate our method on two platforms: 1) using the planner on the test set in simulation and 2) using the planner on a real Franka Panda robot in cluttered environments that also contains unseen objects to study generalization.

V. Results

We will first discuss the effect of the different components in our model, followed by the performance of our method in Section. V-A. We will then discuss the effectiveness of different types of feedback in Sec. V-C and our generalization experiments in Sec. V-D. We also show failures in Sec. V-E.

A. Goal Reaching with Language

First, we test how our method can be leveraged to reach positions given directly as (x,y) to the planner, when the robot is stuck in a local minimum. We evaluate on the test dataset where the planner was 100% successful and also on the *hard* set where the planner had 0% success. Visualizations of some of these hard environments can be found in appendix B. With just a single language correction (*Single-correction*) we can improve success rate from 0% to 81% in the *hard*-set of problems, which also brings our success rate to 98% on the full test set. With another language correction (*Two-corrections*, we get our success rate to 93% and 99% in the *hard* and test sets respectively. In this way, we see that minimal human input can bring the overall reliability of the system to an impressive level.

We additionally tested our network's ability to understand language by starting the robot at the initial position and specifying the goal solely via the language string (*Goal-as-Language*). In this setting, we do not give the planner access to the desired (x,y) position and as such the success rate drops to 65% on the full set. However, we see that in problems where the original planner failed to reach with access to (x,y) (*hard* set), we see that our network is able to succeed in 29% of the set without requiring access to (x,y). Through the results in Table. I, we can see that the $\mathcal{C}_{\mathcal{L}}$ model pulls the robot in most situations with one or two

corrections. A full list of environments, corrections provided, cost maps produced and trajectories can be found on the website sites.google.com/view/language-costs.

Model	Success Rate	
	Hard	Solvable + Hard
Planner	0%	94%
Single-correction	81%	98%
Two-corrections	93%	99%
Goal-as-Language	29%	65%

TABLE I: Performance on Goal Reaching

Category	Success on Solvable	Object Reference
No-Language	4%	0%
No-Vision	33%	0%
No-U-Net-Encoder	33%	0%
Ours(No-Mask)	58%	37%
Cost and Mask (Ours)	**69%**	**52%**

TABLE II: Effect of various model ablations on performance, when given a goal language instruction and no Euclidean distance cost. We see that our proposed method which predicts both a cost and valid-area mask outperforms the alternatives. Figure 13 in the appendix shows the rate convergence to the goal. It can be seen that the difference between performance of the No-mask and our model is on the medium and long trajectories.

B. Ablation Experiments

To evaluate the effect of different components of our model, we run our method in simulation with our *solvable* test set of situations (independent and identically distributed with the training data). Again, we specify the goal only as a language instruction but do not give the planner the (x, y) position of the goal. This enables us to quantify the performance of the mapping from language string to planner success without any bias from a goal cost.

We first disable the language module so that our network doesn't take any language input (*No-Language*). This is an under-specified system as the network does not know what the user feedback is and hence the success rate is only 4% as seen in table. II. This ablation shows that our dataset is not biased and it indeed requires language input for success.

We then removed the vision input to the network (*No-Vision*), so both CLIP ViT and the U-Net encoder do not get the environment image or the location of the robot. This is done to test if simply given a language instruction, how well does exhibiting an average behavior do and the success rate is only at 33%. This success rate includes only very basic commands, like "go left", "go down", "speed up", et cetera; without vision, the system cannot accomplish any task that refers to an object. When we remove our U-Net encoder (*No-U-Net-Encoder*) and only use input to CLIP, the network does not do any better than (*No-Vision*), implying that the 512 vision embedding from CLIP is not sufficient to encode our task specific environments.

Finally, we remove only the trajectory mask $\mathcal{M}$, and only use $\mathcal{C}$ as cost in *No-Mask*. We see that this brings up the success rate to 58% but adding the trajectory mask get our method to 69% on the test dataset.

C. Performance by Instruction Type

In this section, we analyze our model's performance when given specific types of instruction. We trained on five spatial object dependent tasks-[Above, Below, Left of, Right of, Stay Away], two spatial robot object dependent tasks-[Behind, In front of], 4 directional spatial robot dependent tasks and 2 velocity tasks (fast, slow). Table III shows a performance breakdown on each of these tasks. We evaluate on the solvable set of successful examples and split the trajectories by the total length of the ground-truth trajectory to analyze whether performance is dependant on the trajectory length. Specifically, paths of planning time steps < 40 were short, $40-60$ medium, and > 60 were long. There were 304 short examples, 131 medium examples, and 108 long examples. When running these evaluations, we do not give access of (x, y) goal positions to the planner and only give a language string specifying where the robot needs to go. The overall success rate was 69%; we see that specifically very local corrections such as "go faster" were always successful. Fig. 5 shows the interaction between environment, cost prediction, and mask prediction for various goal instructions.

Type of Feedback	Success Rate on Solvable Set			
	All	Short	Medium	Long
Above	77%	86%	83%	40%
Below	66%	70%	77%	46%
Left of	45%	82%	33%	12%
Right of	49%	92%	47%	17%
Behind	20%	17%	26%	18%
In front of	76%	83%	50%	-
Positional(↕↔)	100%	100%	-	-
Velocity	100%	-	-	-
Stay Away	95%	-	-	-

TABLE III: How effective is the language cost prediction module at navigating to various types of goals? We look at problems of varying levels of difficulty. The types are partitioned by horizontal lines based on categories described in Sec V-C

D. Generalization Experiments

The CLIP embeddings used in our model provide a strong basis for language generalization, as seen in previous work [40, 35]. We performed additional set of experiments to show how our models preserve this generalization ability, making them more broadly useful despite the small amount of training data on only four objects. We evaluate on the real robot for these results, where the environment also contains unseen objects in clutter. The results of these generalization experiments are in Table IV, which shows how our approach can scale to a wide range of problems.

To study generalization to unseen language instructions, we referred to the objects with non-templated phrases and synonymous object names that were not part of the training

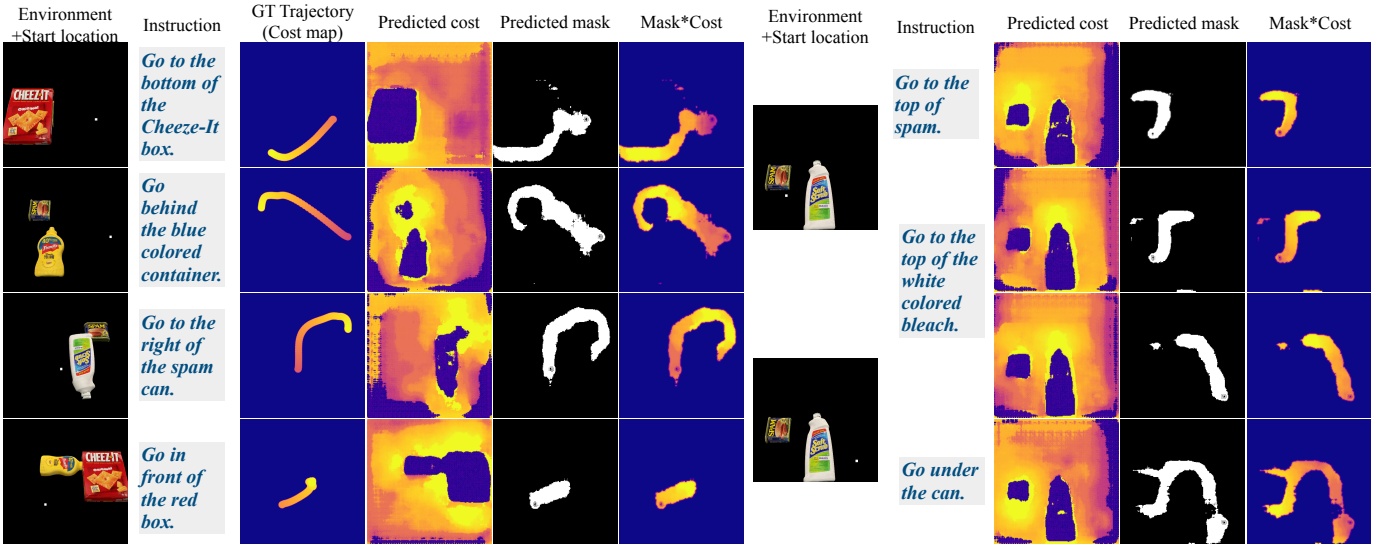


Fig. 5: The predicted cost map and generated trajectory together create a masked path for the robot to follow. The costs could also be used in isolation from the mask, though this results in worse performance. (Right) The model is sensitive to the environment, the start location of the robot and the instruction. Here, we show the costmap and masked costmap change as we hold other variables constant.

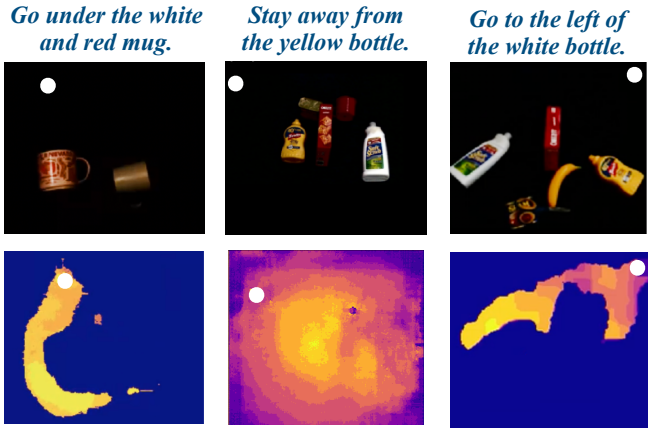
vocabulary, in 20 different setups and found that our method was able to successfully complete the task in 17 of them. Fig. 9 shows some examples of diverse natural-language sentences used to control the robot. We also tested our method with language instructions referring to 10 unseen objects and our method worked on *cups(red, orange), plate, fork, ketchup bottle* and failed on *screwdriver, candle, book, banana, pepper can, pen*. When any of the unseen objects were in the background (clutter) and language instructions were in reference to seen objects, our method succeeded always even when the objects were placed in orientations that were not seen during training as shown in Fig. 6. Our training data only contained scenes with two of four objects in poses chosen from a fixed set of 4 orientations while the evaluations we did on the real robot contained many objects and seen objects in different orientations.

Generalization type	Task Success
New language instructions	17/20
Reference to unseen objects	4/10
New objects in background; clutter	15/15
Unseen poses for known objects	14/15

TABLE IV: Performance on generalization experiments. Our approach can generalize to new objects, language instructions, and to clutter that was not present in the training data set.

E. Failures

Most failures of the correction policy are either due to a discontinuity in the trajectory mask generated or due to some few-off pixels in the cost map along the trajectory. Examples of these can be seen in Fig 10. The figure also describes other curious scenarios. First, in the absence of an observable path $\mathcal{C}_\mathcal{L}$ tries to find a path from the edges of the frame. In the case of environments with two instances of the same object the model generates two distinct paths to both the objects. This



a) Objects b) Number of objects c) Object poses

Fig. 6: Visual Generalization: Visually the model generalized to unseen objects in the scene, to a different number of objects in the scene (as compared to 2 at training time) and to new poses of objects.

is also true in cases when there are two objects and there is ambiguity in the instruction as seen in Fig 10.

F. Discussion

In addition to generalization, our approach has the advantages of compositionality over other means of providing feedback to improve robot behavior. We can specify multiple constraints at execution time and combine them in the $\mathcal{C}_\mathcal{L}$ term. These can either be provided at once or at different times through the trial. We provide demonstrations showing a robot's behavior at combining velocity cost with goal reaching costs and with stay away cost while the robot is trying to reach a goal on our website sites.google.com/view/language-costs

Issuing commands at different times is also a powerful and intuitive way to control the robot. For example, in Fig. 7

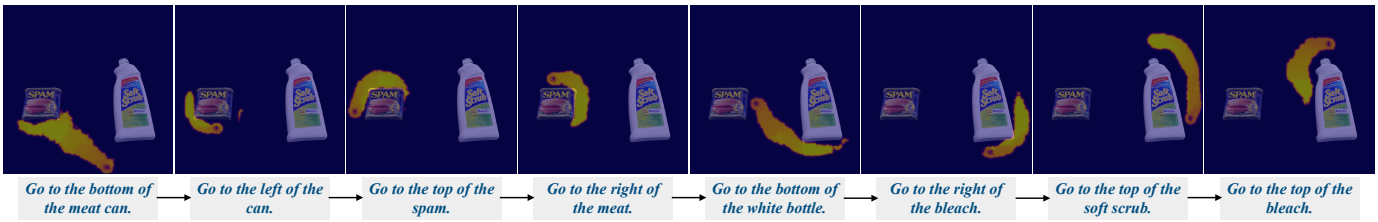


Fig. 7: Compositionality in time. Our approach allows the user to specify multiple different cost functions at different points in time, letting them guide the robot around an intended trajectory with language.

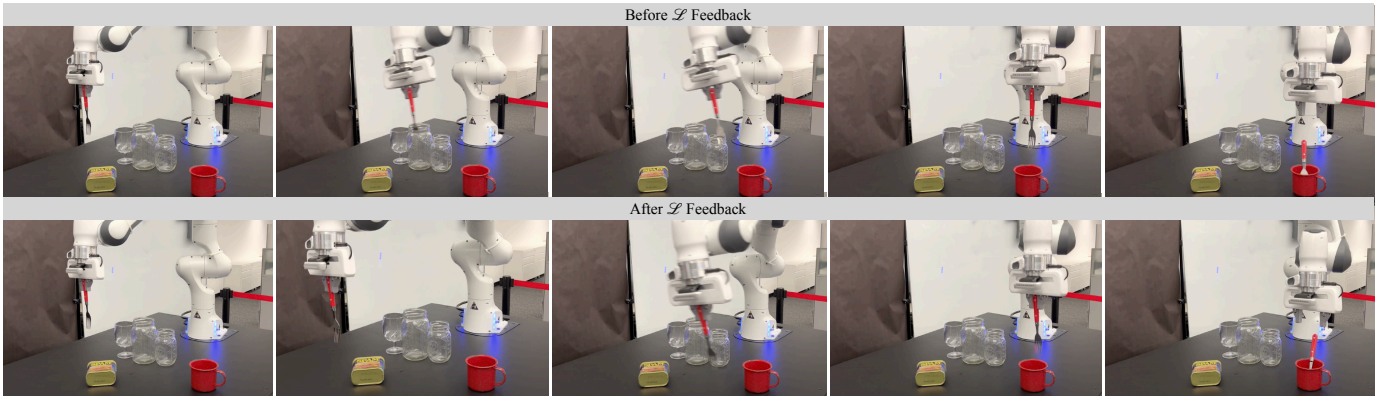


Fig. 8: An example using our framework to tell a robot to avoid a set of fragile, glass objects while performing a pick and place task. Before, the robot moves dangerously close to the glass containers; after "go through the bottom of spam", it avoids them and maintains a safe distance.

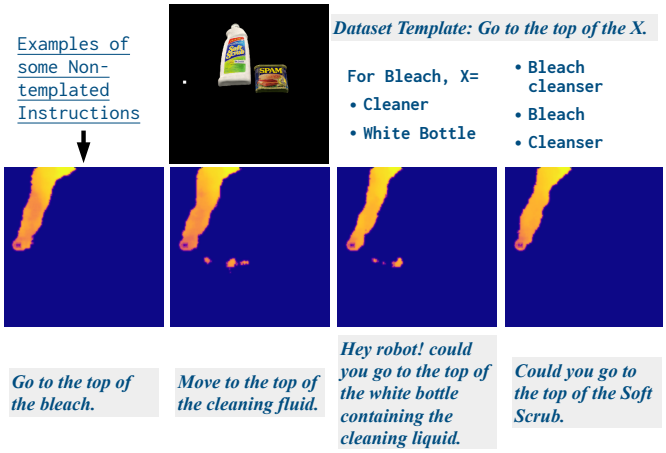


Fig. 9: The language correction module is trained with templated language instructions. For example, to teach the robot to go over an object, we use the template "Go to the top of the X," where X is the name of an object that can be referred to in several ways. However, in spite of this, our model can generalize to a wide variety of different types of instructions, as shown above.

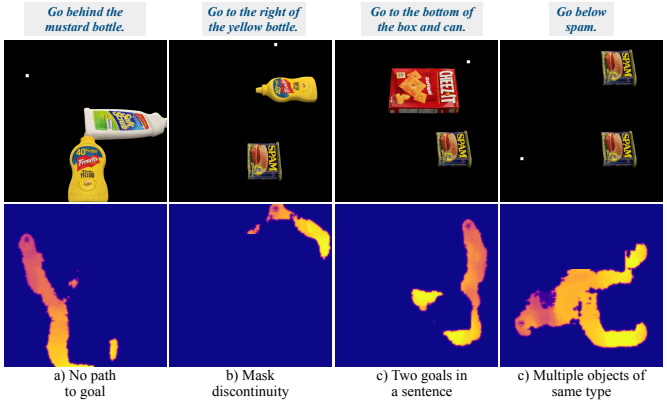


Fig. 10: Some failures of our approach is shown here. Our method can produce masks that go outside the bounds of the image (left) or discontinuous masks in some instances. Our method also cannot distinguish between two objects of the same type as shown in the right.

we see the robot being asked to go to different locations around objects sequentially. Chaining corrections makes possible specifying procedures, and correcting trajectories that require more than a single correction to be corrected. It might also provide a way to generate data to learn more complex behaviors.

One useful feature of our approach is the ability to correct behaviors across multiple environments at once. Take the example in Fig. 11: "go to the left of the bleach." This correction can be applied in every environment, even if the robot is moving to different goals, with no additional effort on the part of the user. To make such a correction with other means such as a joystick or kinesthetic feedback would require considerable time and effort.

Importantly, corrections do not always need to be provided *only* in the case of a planner failure, but can be used by the user to modify the plan based on their *preference*. In Figure 8

we see a human providing a correction to steer the robot away from fragile objects. This correction is applied to the task of placing an object in the mug.

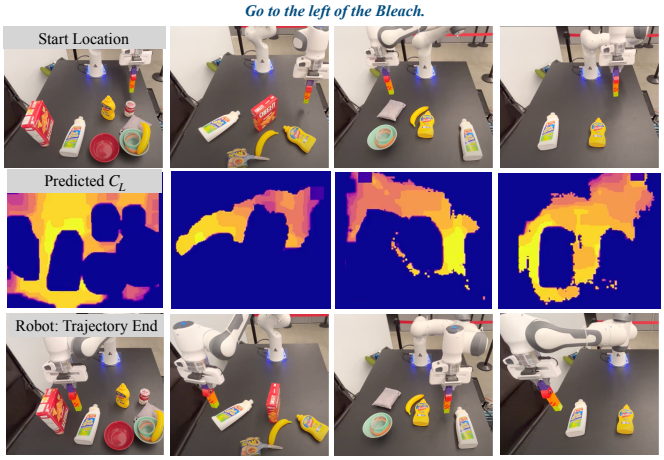


Fig. 11: Providing a single language command in multiple environments. One significant advantage of using language for correcting robot behaviors is how it allows us to issue the same correction in many environments at once.

VI. Related Work

This work builds upon multiple threads of related work.

1) Natural Language for Robot Behavior Correction: Correcting robot behavior using language has been studied for robots that ask for help [38], understand language corrections [10, 12], and use language to disambiguate joystick corrections [21]. Broad et al. [10] use a grounding model based on a Distributed Correspondence Graph [18] to ground corrections, which limits grounding language to a hand-engineered set of optimization constraints. In contrast, we directly learn to predict cost maps using a neural network, side-stepping the potentially laborious constraint design process. Co-Reyes et al. [12] learn a policy that accepts corrections in addition to an instruction, however it requires training with corrections at training time which makes it sample inefficient as number of tasks scale. Further, it does not permit decoupling the notions of a goal and a trajectory to said goal when issuing a correction. Karamcheti et al. [21] use language to disambiguate underspecified joystick corrections by learning a mapping to robot joint space. This suffers from the same limitations as language-free joystick [36, 33] or kinesthetic [6] corrections, requiring undivided user attention. Furthermore, their language grounding model is based on a nearest-neighbour lookup, which does not generalize to new environments and tasks.

2) Language for Robot Task Specification: Natural language has been extensively studied as a means for task specification or instruction following in robotics [37, 22, 15, 4, 28, 44, 8, 31, 42]. Mapping language to symbolic plans [37, 22, 24, 4], has enabled following instructions by invoking a set of pre-specified skills or motion primitives. Recently, instruction following has been studied in robotics by mapping instructions and raw visual observations to actions using end-to-end representation learning and sim-to-real transfer [8, 2, 35]. All of these works, however, treat language instructions as goals that are fixed during execution. In contrast, by framing language grounding as cost prediction, we enable the use of language to refine robot behavior over time, while still allowing instruction-following as a special use-case.

3) Inferring costs from demonstrations: Our correction model is trained to map observations and language to cost maps, on data consisting of demonstrations or ground-truth cost maps. This is related to Inverse Reinforcement Learning (IRL) [1] that learns to recover reward functions from demonstrations, and has been successfully applied to infer objectives for manipulation motion planners [20]. In contrast, our cost model is conditioned on language and visual observations, which enables re-using the same model with diverse language corrections in a variety of tasks.

4) Value alignment problem: Our method presents an interactive solution to the value alignment problem [43], whereby the cost function provided to the robot is not reflective of the true task that the user has in mind. Our language corrections enable interactively updating the cost to better reflect the task. This problem has been also addressed by learning to predict true rewards given observed rewards across environments and tasks [16], and by learning from physical interactions with humans [5].

VII. Conclusion

In conclusion, we proposed a framework to integrate human provided feedback in natural language to update a robot's planning cost applied to situations when the planner fails. This is done by modelling cost associated with the language instruction $\mathcal{C}_{\mathcal{L}}$ conditioned on the language feedback $\mathcal{L}$. The $\mathcal{C}_{\mathcal{L}}$ can be used in conjunction with the motion planner's existing costs. We train our model on data generated via simulation and evaluate the performance of the model in various out of distribution settings in the real world involving non-templated natural user commands, cluttered scenes, new poses and types of objects.

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.

[2] Peter Anderson, Ayush Shrivastava, Joanne Truong, Arjun Majumdar, Devi Parikh, Dhruv Batra, and Stefan Lee. Sim-to-real transfer for vision-and-language navigation. In *CoRL*, 2020.

[3] Jacob Andreas and Dan Klein. Alignment-based compositional semantics for instruction following. In *EMNLP*, 2015.

[4] Dilip Arumugam, Siddharth Karamcheti, Nakul Gopalan, Lawson LS Wong, and Stefanie Tellex. Accurately and efficiently interpreting human-robot instructions of varying granularities. In *RSS*, 2017.

[5] Andrea Bajcsy, Dylan P. Losey, Marcia Kilchenman O'Malley, and Anca D. Dragan. Learning robot objectives from physical human interaction. In *CoRL*, 2017.

[6] Andrea Bajcsy, Dylan P. Losey, Marcia K. O'Malley, and Anca D. Dragan. Learning from physical human corrections, one feature at a time. In *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 141–149, 2018.

[7] Mohak Bhardwaj, Balakumar Sundaralingam, Arsalan Mousavian, Nathan D Ratliff, Dieter Fox, Fabio Ramos, and Byron Boots. Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation. In *5th Annual Conference on Robot Learning*, 2021.

[8] Valts Blukis, Yannick Terme, Eyvind Niklasson, Ross A. Knepper, and Yoav Artzi. Learning to map natural language instructions to physical quadcopter control using simulated flight. In *CoRL*, 2019.

[9] Stuart A Bowyer, Brian L Davies, and Ferdinando Rodriguez y Baena. Active constraints/virtual fixtures: A survey. *IEEE Transactions on Robotics*, 30(1):138–157, 2013.

[10] Alexander Broad, Jacob Arkin, Nathan Ratliff, Thomas Howard, Brenna Argall, and Distributed Correspondence Graph. Towards real-time natural language corrections for assistive robots. In *RSS Workshop on Model Learning for Human-Robot Communication*, 2016.

[11] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *2015 international conference on advanced robotics (ICAR)*, pages 510–517. IEEE, 2015.

[12] John D Co-Reyes, Abhishek Gupta, Suvansh Sanjeev, Nick Altieri, Jacob Andreas, John DeNero, Pieter Abbeel, and Sergey Levine. Guiding policies with language via meta-learning. In *ICLR*, 2018.

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

[14] Anca D. Dragan and Siddhartha S. Srinivasa. A policy-blending formalism for shared control. *The International Journal of Robotics Research*, 32:790 – 805, 2013.

[15] Felix Duvallet, Thomas Kollar, and Anthony Stentz. Imitation learning for natural language direction following through unknown environments. *2013 IEEE International Conference on Robotics and Automation*, pages 1047–1053, 2013.

[16] Dylan Hadfield-Menell, Smitha Milli, P. Abbeel, Stuart J. Russell, and Anca D. Dragan. Inverse reward design. In *NIPS*, 2017.

[17] Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 1990.

[18] Thomas M Howard, Stefanie Tellex, and Nicholas Roy. A natural language planner interface for mobile manipulators. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6652–6659. IEEE, 2014.

[19] Ashesh Jain, Shikhar Sharma, Thorsten Joachims, and Ashutosh Saxena. Learning preferences for manipulation tasks from online coactive feedback. *The International Journal of Robotics Research*, 34:1296 – 1313, 2015.

[20] Mrinal Kalakrishnan, Peter Pastor, Ludovic Righetti, and Stefan Schaal. Learning objective functions for manipulation. In *ICRA*, 2013.

[21] Siddharth Karamcheti, Megha Srivastava, Percy Liang, and Dorsa Sadigh. Lila: Language-informed latent actions. In *CoRL*, 2021.

[22] Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. Learning to parse natural language commands to a robot control system. In *ISER*, 2012.

[23] Dipendra Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. Tell me dave: Context-sensitive grounding of natural language to mobile manipulation instructions. In *Robotics: Science and Systems (RSS)*, 2014.

[24] Dipendra K Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. Tell me dave: Context-sensitive grounding of natural language to mobile manipulation instructions. In *RSS*, 2014.

[25] Raymond J Mooney. Learning to connect language and perception. In *AAAI*, pages 1598–1601, 2008.

[26] Nathan Morrical, Jonathan Tremblay, Yunzhi Lin, Stephen Tyree, Stan Birchfield, Valerio Pascucci, and Ingo Wald. Nvisii: A scriptable tool for photorealistic image generation, 2021.

[27] Arne Muxfeldt, Jan-Henrik Kluth, and Daniel Kubus. Kinesthetic teaching in assembly operations - a user study. In *SIMPAR*, 2014.

[28] Daniel Nyga, Subhro Roy, Rohan Paul, Daehyung Park, Mihai Pomarlan, Michael Beetz, and Nicholas Roy. Grounding robot plans from natural language instructions with incomplete world knowledge. In *CoRL*, 2018.

[29] Aishwarya Padmakumar, Peter Stone, and Raymond J. Mooney. Learning a policy for opportunistic active learning. In *EMNLP*, 2018.

[30] Rohan Paul, Jacob Arkin, Nicholas Roy, and Thomas M Howard. Efficient grounding of abstract spatial concepts for natural language interaction with robot manipulators. 2016.

[31] Chris Paxton, Yonatan Bisk, Jesse Thomason, Arunkumar Byravan, and Dieter Fox. Prospection: Interpretable plans from language by predicting the future. In *ICRA*, 2019.

[32] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, 2021.

[33] Daniel Rakita, Bilge Mutlu, and Michael Gleicher. An

autonomous dynamic camera method for effective remote teleoperation. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, pages 325–333, 2018.

[34] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

[35] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *Conference on Robot Learning*, pages 894–906. PMLR, 2022.

[36] Jonathan Spencer, Sanjiban Choudhury, Matt Barnes, Matthew Schmittle, Mung Chiang, Peter Ramadge, and Siddhartha Srinivasa. Learning from Interventions: Human-robot interaction as both explicit and implicit feedback. In *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, July 2020. doi: 10.15607/RSS.2020.XVI.055.

[37] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R. Walter, Ashis Gopal Banerjee, Seth J. Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, 2011.

[38] Stefanie Tellex, Ross Knepper, Adrian Li, Daniela Rus, and Nicholas Roy. Asking for help using inverse semantics. In *RSS*, 2014.

[39] Stefanie Tellex, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek. Robots that use language. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1):25–55, 2020. doi: 10.1146/annurev-control-101119-071628. URL https://doi.org/10.1146/annurev-control-101119-071628.

[40] Jesse Thomason, Mohit Shridhar, Yonatan Bisk, Chris Paxton, and Luke Zettlemoyer. Language grounding with 3d objects. In *Conference on Robot Learning*, pages 1691–1701. PMLR, 2022.

[41] Sagar Gubbi Venkatesh, Raviteja Upadrashta, and Bharadwaj Amrutur. Translating natural language instructions to computer programs for robot manipulation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1919–1926, 2021. doi: 10.1109/IROS51168.2021.9636342.

[42] Thomas Victor Ilyevsky, Jared Sigurd Johansen, and Jeffrey Mark Siskind. Talk the talk and walk the walk: Dialogue-driven navigation in unknown indoor environments. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4903–4910, 2021. doi: 10.1109/IROS51168.2021.9636548.

[43] Wendell Wallach and Colin Allen. Moral machines: Teaching robots right from wrong. 2008.

[44] Edward C Williams, Nakul Gopalan, Mine Rhee, and Stefanie Tellex. Learning to parse natural language to grounded reward functions with weak supervision. In *ICRA*, 2018.

[45] Tian Xia, Simon Léonard, Isha Kandaswamy, Amy Blank, Louis L Whitcomb, and Peter Kazanzides. Model-based telerobotic control with virtual fixtures for satellite servicing tasks. In *2013 IEEE International Conference on Robotics and Automation*, pages 1479–1484. IEEE, 2013.

APPENDIX

A. Planner

We use STORM [7] as the planner which computes an action leveraging sampling based optimization to optimize over costs. The base cost $\mathcal{C}_{\mathcal{B}}$ for the 2D simulation robot contains the following terms:

$$\mathcal{C}_{\text{joint}}(s_t) = \begin{cases} ||s_t - s_{min}||_2 & \text{if } s_t < s_{min} \\ ||s_{max} - s_t||_2 & \text{else if } s_t > s_{max} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{C}_{collision}(q_t) = Coll(q_t, o^h)$$

where $Coll(\cdot)$ checks for collisions between the robot position and the image o^h using a binary mask.

When using the planner on the Franka Panda robot, we use the cost terms described in [7] with the following changes:

1) We only check for environment collisions with the table. We don't check for collisions with objects and rely on $\mathcal{C}_{\mathcal{L}}$ to ensure the ensure that the trajectory taken by the robot is not in collision.
2) We add a cost to constrain the position along z-axis and 3D orientation of the gripper during execution to a default value that's close to the table.

Across all instances of the planner, we use 500 particles and a horizon of 30 timesteps.

B. Hard environments

Some examples of environments where the planner fails can be seen in Figure 12. An MPC model minimizing the $\mathcal{C}_{\mathcal{T}}$ from the start location to goal gets stuck in hard to escape local-minima solutions. The robot is required to take several steps along a trajectory of increasing $\mathcal{C}_{\mathcal{T}}$ in order to reach a point starting where the robot can resume minimizing the specified $\mathcal{C}_{\mathcal{T}}$ to reach the true goal. It is these inflection points that the natural language feedback, $\mathcal{L}$, helps point the robot to.

Fig. 12: Hard Environments: Examples of some of the environments in the hard dataset. The White dot is the start location of the robot and the green dot is the goal location. The planner takes paths that lead it down to bad-local minima in these environments.

C. Convergence to Goal : Analysis

A natural question to ask is what can one do when the correction module itself fails and afterall, it is also a model not immune to failures. Here we understand when the correction module fails. We group trajectories into easy medium and hard based on the length of the trajectories. It can be seen that corrections with longer trajectories are much worse than corrections with shorter trajectories. The main insight is that despite having a limited correcting ability one can still use it to make simple modifications at once or sequentially to correct behaviors.

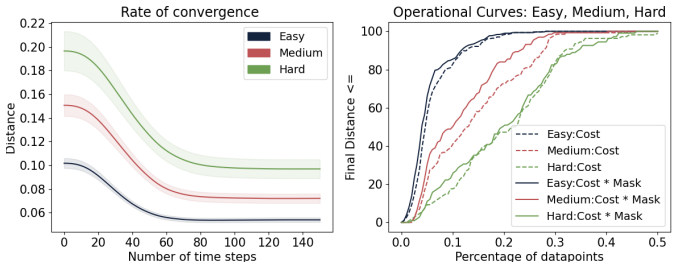


Fig. 13: Convergence to Goal Analysis: a) As discussed in the results section the planner does better at short-horizon tasks as compared to long horizon tasks. The interesting aspect of the model is that even for long-horizon tasks, the first part of the trajectory does move in a direction where the goal is minimised for several steps. Even a model for $\mathcal{C}_{\mathcal{L}}$ with varying performance across short and long horizon corrections can still do well on introducing corrections that improve planner performance b) The advantage of generating a mask $\mathcal{M}$ over the cost is most evident for medium to long trajectories.