DEMONSTRATE: Zero-shot Language to Robotic Control via Multi-task Demonstration Learning

Rahel Rickenbach*, Bruce Lee†, René Zurbrügg*, Carmen Amo Alonso‡, and Melanie N. Zeilinger*
*ETH Zurich, †University of Pennsylvania,‡Stanford University
{rrahel,zrene,mzeilinger}@ethz.ch, brucele@seas.upenn.edu, camoalon@stanford.edu

***Abstract*—The integration of large language models (LLMs) with control systems has demonstrated significant potential in various settings, such as task completion with a robotic manipulator. A main reason for this success is the ability of LLMs to perform in-context learning, which, however, strongly relies on the design of task examples, closely related to the target tasks. Consequently, employing LLMs to formulate optimal control problems often requires task examples that contain explicit mathematical expressions, designed by trained engineers. Furthermore, there is often no principled way to evaluate for hallucination before task execution. To address these challenges, we propose DEMONSTRATE, a novel methodology that avoids the use of LLMs for complex optimization problem generations, and instead only relies on the embedding representations of task descriptions. To do this, we leverage tools from inverse optimal control to replace in-context prompt examples with task demonstrations, as well as the concept of multitask learning, which ensures target and example task similarity by construction. Given that hardware demonstrations can easily be collected using teleoperation or guidance of the robot, our approach reduces the reliance on engineering expertise for designing in-context examples. Furthermore, the enforced multitask structure enables learning from few demonstrations and assessment of hallucinations prior to task execution. We demonstrate the effectiveness of our method through simulation and hardware experiments involving a robotic arm tasked with tabletop manipulation.**

I. Introduction

Optimization-based control strategies, such as model predictive control (MPC), have demonstrated great potential in robotics for solving complex tasks in constrained environments. However, their design can be challenging and require extensive domain knowledge. Recent studies have addressed this issue by leveraging large language models (LLMs) to reduce the manual tuning effort and enable human-robot communication through natural language. One approach is to directly use a pre-trained LLM to determine low-level robotic actions from a natural language prompt [1, 2, 3]. This approach typically requires extensive fine-tuning since LLMs often struggle with spatial reasoning tasks [4]. Moreover, it lacks the constraint satisfaction guarantees of advanced control approaches. Another approach is to leverage LLMs for optimal controller design, where the LLM formulates all or some elements of the optimal control problem (OCP), either as code scripts [5, 6, 7] or mathematical expressions [8, 9, 10]. While these approaches offer a more principled path toward robotic control via natural language commands, they are hindered by current limitations of LLMs. In particular, given the high sensitivity of LLMs to the prompt, current approaches require highly tuned in-context examples from skilled engineers, which defeats the purpose of using language to ease communication with robots. In this paper, we propose DEMONSTRATE, a novel way to interface LLMs and controllers that mitigates this issue. To do so, we build upon the NARRATE pipeline [11], where an LLM is used to map a language utterance to a set of objective and constraint functions for an OCP. However, rather than assuming access to highly optimized example prompts, as done in [11], we assume access to a collection of robotic demonstrations for various low-level sub-tasks. We then achieve comparable empirical success-rates to [11], by using these demonstrations to form a mapping from embeddings of natural language descriptions for the sub-tasks to the functions describing the OCP. Doing so frees the LLM from the complex task of directly generating the OCP, and provides a novel quantitative way to assess hallucinations before the task is executed. Moreover, since only embedding representations are required, computation of the mapping output is significantly more efficient than relying on LLM generations. This is achieved via the following two key contributions.

- First, we show how to leverage approaches from the field of inverse reinforcement learning (IRL) and inverse optimal control (IOC) [12], to learn the objective and the constraints from the provided manual demonstrations.
- Second, we build upon the observation that useful in-context examples are relevant to the task at hand and leverage this idea through the concept of multi-task representation learning [13, 14].

In particular, we employ multi-task representation learning to make task-similarity, which is often not well-defined and handled by the intuition of an expert, systematic and ensure similarity of target and example tasks by construction. Furthermore, we use the available demonstrations to extract compressed features of objective and constraint functions and map the embedding space to this shared feature representation. In this way, our architecture can perform zero-shot generalization to new tasks described in natural language.

II. Related Work

The use of large language models (LLMs) for control has been successfully applied across various domains, ranging from autonomous driving to robotic manipulation and actuation [1, 15, 11]. To achieve optimal performance, typically one of two strategies are applied: adaptive reasoning or precise

prompting. In adaptive reasoning, as demonstrated in [15], the LLM is used not only to break down tasks into sub-tasks and design optimization-based controllers but also to provide corrective feedback during execution. Alternatively, precise prompting involves crafting specific, task-relevant prompts, as seen in [11, 1, 16]. These prompts can be categorized into two types: visual [16] and textual [11, 1]. However, creating effective textual prompts often requires expertise from skilled engineers or the integration of a well-tuned controller. The quality of these prompts has been shown to significantly impact the performance of LLM-based controllers [17, 18, 19]. The sensitivity of LLM performance to prompt design has been mitigated through various prompt optimization techniques. For instance, automated methods like autoprompt [20, 21] and reinforcement learning (RL)-based prompt optimization [22, 23] have been developed to refine prompts systematically. In cases where prompts are manually designed [1, 24], the issue of response sensitivity to user commands has been addressed using uncertainty quantification techniques [25]. These methods classify user commands based on their clarity, enabling the LLM to assess the reliability of its responses. Furthermore, such uncertainty quantification can empower the LLM to proactively seek assistance when faced with ambiguous or unclear inputs [26].

III. Problem Statement

In this section, we present the problem of interest and the proposed concept, which is then detailed in Section IV.

A. Problem Setup

Given a robotic system, the goal is to complete a task $\mathcal{P}$ described in terms of a natural language command, denoted $\ell_{\mathcal{P}}$. A prominent approach in the literature is to convert $\ell_{\mathcal{P}}$ into multiple target sub-tasks τ, described with a natural language command ℓ_τ, which allow for the translation into an OCP, solved via an MPC [27] at each time step k:

$$\min_{\substack{x_0,\dots,x_N \\ u_0,\dots,u_{N-1}}} \quad c_\tau(x_0,\dots,x_N,u_0,\dots,u_{N-1}) \tag{1a}$$

$$s.t. \quad x_{i+1} = f(x_i,u_i), \quad i=1,\dots N, \tag{1b}$$

$$x_0 = x(k), \tag{1c}$$

$$(x_i,u_i) \in \mathcal{X}_\tau, \quad i=1,\dots N. \tag{1d}$$

Here, $x(k) \in \mathbb{R}^{n_x}$ is the state and $u(k) \in \mathbb{R}^{n_u}$ is the control input, equation (1b) captures the dynamics of the robot, and $c_\tau(\cdot)$ and $\mathcal{X}_\tau$ represent the cost and constraint set for task τ, respectively. While LLMs have shown promising performances in the division of complex tasks into smaller subtasks, in this paper we address the question:
How can we systematically and reliably map a given natural language utterance ℓ_τ to the OCP representation in (1) with the appropriate c_τ and $\mathcal{X}_\tau$?

Recent work (NARRATE) [11] addresses this problem by relying solely on the capabilities of current LLMs to map language utterances describing the task τ into their corresponding mathematical expressions, c_τ and $\mathcal{X}_\tau$. To do this, a language model L takes as input both (i) the language command ℓ_τ, and (ii) a system prompt p_τ, and outputs a symbolic cost function and a set of constraints: $(c_\tau, \mathcal{X}_\tau) = L(\ell_\tau, p_\tau)$. The system prompt p_τ is a string of text that consists of T in-context examples that associate a natural language prompt ℓ_t with the cost c_t and constraint set $\mathcal{X}_t$ for each example sub-task $t = 1,\dots,T$; i.e., for each example sub-task t, the prompt p_τ contains a 3-tuple of the form $(\ell_t, c_t, \mathcal{X}_t)$. Three important issues arise with this approach:

1) Access to adequate in-context examples requires technical expertise to appropriately design the cost and constraints functions, which is often not available.
2) Significant sensitivity to the prompt: slight modification in the prompt can lead to very different success rates.
3) There is no principled way of assessing hallucinations of the language model before task execution.

In this paper, we study the problem of how to set up OCP (1) as specified through natural language commands, when no direct in-context examples are available. Instead, we assume access to demonstrations from either a human user or an existing controller that successfully completes T sub-tasks, each paired with a natural language description l_t (with $t = 1,\dots,T$). We show how learning from demonstrations of various sub-tasks paired with pre-trained language embedding models and enforcing a particular structure of OCP (1) bypasses the need for in-context examples of OCPs corresponding to a language description of a sub-task and allows for hallucination detection prior to task execution.

B. Approach

To address the aforementioned challenges, our approach relies on two innovations. First, we leverage tools that allow for the construction of an OCP from successful demonstrations of sub-task examples. Second, we systematically ensure that sub-task examples are closely related to potential target sub-tasks, by sharing a common mathematical representation for the OCP. More specifically, we harness the following concepts.

a) Learning from Demonstrations: Demonstrations consist of state and action measurements over the course of a trajectory, i.e., $(\mathbf{x},\mathbf{u}) := (x_0,\dots,x_N,u_0,\dots,u_{N-1})$. Given a collection of (potentially suboptimal) demonstrations of an example sub-task t, they can be used to learn an objective function c_t, e.g., with so-called entropy maximization methods [28, 29], as well as the constraints space $\mathcal{X}_t$, via the construction of constraint violating input sequences [30, 31].

b) Multitask Embedding Mapping: Good choices of in-context example sub-tasks (or demonstrations) carry similarity with potential target tasks. We use this concept in two ways. First, we take advantage of the resulting structure of the language description representation of these sub-tasks in the embedding space. Second, we define a task space $\mathcal{S}$ such that for all $\bar{t} \in \mathcal{S}$, with $\bar{t}$ representing either a target sub-task τ or an example sub-task t, $c_{\bar{t}}$ and $\mathcal{X}_{\bar{t}}$ enjoy a specific mathematical structure that can be parameterized via feature vectors and enforces structural similarities between example and target sub-tasks.

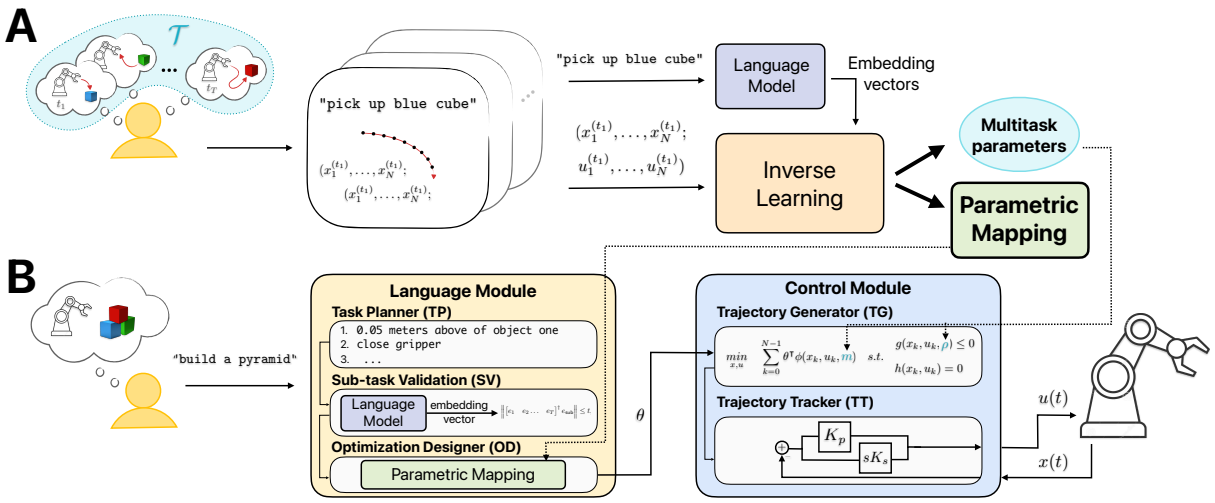

Fig. 1: **A. Offline pipeline:** (a) A user provides sub-task descriptions together with trajectory demonstrations; (b) the LLM computes the embedding vectors for the sub-task descriptions; (c) the demonstrations and the embedding vectors are used to learn (i) a *parametric mapping*, and (ii) the multitask parameters, both used in the online pipeline. **B. Online pipeline:** Architecture of [11], with an added SV Module and modified OD. The LLM is used to find the embedding representation of the language commands from TP, which are fed into the *parametric map* to compute the feature vector used in the TG block.

DEMONSTRATE proposes an interface between LLMs and optimal controllers that builds on and augments the existing architecture of [11]. The resulting pipeline is shown in Fig. 1. By using a learning-from-demonstrations approach, the user is able to directly provide example sub-task descriptions together with demonstrations, removing the need for technical expertise to create prompt examples in mathematical terms. Moreover, the multitask approach allows for enhanced robustness by explicitly leveraging the structure of sub-tasks that share a common task space $\mathcal{S}$ and accordingly, common features describing their representing OCPs. By only altering the parametrization of these feature vectors while keeping the remaining structure of the OCP fixed, we ensure a common representation of example and target sub-tasks. Therefore, it becomes possible to directly learn a *parametric mapping* $\mathcal{M}$ between the embedding vectors of the natural language commands and the parametrization of feature vectors describing the task. Hence, generalization to unseen tasks in $\mathcal{S}$ is achieved online via this trained map, alleviating the LLM from complex symbolic function creation. During generation, the *parametric mapping* $\mathcal{M}$ is employed to generate new feature combining parameter vectors of the output sub-task, which, are guaranteed to fall within the pre-defined task feature space, thus mitigating the risk for LLM hallucinations. Additionally, this structure can be used as a correctness assessment by evaluating the similarity between the embedding vector of the generated target and the available example sub-tasks.

IV. Multi-task Demonstration Learning for Language to Control

In this section, we provide a detailed description for the different blocks in the proposed pipeline (Fig. 1). The components of the offline pipeline are discussed in the first subsection, while the adaptation of the online pipeline from [11] is discussed in the subsequent subsection. Further details on the implemented approaches in the offline pipeline are presented in the Appendix A.

A. Offline Multitask Learning From Demonstrations

The proposed offline pipeline has two main steps: (1) proper design and collection of demonstrations, and (2) learning from collected demonstrations.

1) Demonstrations design and data collection:

a) Demonstrations design and multitask setting: Inspired by multi-task learning approaches [13, 14], example and target sub-tasks are assumed to share a common mathematical description $\mathcal{S}$. In particular, for every sub-task $\bar{t} \in \mathcal{S}$, we assume the cost $c_{\bar{t}}(\cdot)$ and the constraint set $\mathcal{X}_{\bar{t}}$ in (1a) and (1d) are of the form:

$$c_{\bar{t}}(x,u) = \theta_{\bar{t}}^{\intercal}\phi(x,u,m_{\bar{t}}), \tag{2a}$$

$$\mathcal{X}_{\bar{t}} \subseteq \{x,u \; s.t. \; g_j(x,u,\rho_{\bar{t}}) \leq 0, \forall j = 1,\dots,J\}, \tag{2b}$$

for some parameter vectors $\theta_{\bar{t}} \in \mathbb{R}^p$, $m_{\bar{t}} \in \mathbb{R}^{q_m}$, and $\rho_{\bar{t}} \in \mathbb{R}^{q_\rho}$. The maps $g(\cdot), \phi(\cdot) \in C^1$ are design choices and ideally parametrized on the basis of feature vectors that capture both target and example sub-tasks. To avoid the necessity of an expert, examples of $\phi(\cdot)$ and $g(\cdot)$ can be pre-implemented in the pipeline, ranging from tailored collections of parametrized norms and Gaussian Process [32] approximating trigonometric functions for $\phi(\cdot)$, to half-spaces and ellipsoids for $g(\cdot)$. Mathematically, we capture the multitask assumption as follows:

Assumption 1: All demonstration and target tasks belong to $\mathcal{S}$ as defined by the cost and constraint set (2), where $m_{\bar{t}_1} = m_{\bar{t}_2}$ and $\rho_{\bar{t}_1} = \rho_{\bar{t}_2}$ for every pair $\bar{t}_1, \bar{t}_2 \in \mathcal{S}$, i.e., all tasks in $\mathcal{S}$ share the parameter vectors m and ρ. Task-specificity is defined via θ.

b) Data Collection: For each demonstrated sub-task example $t_1,\dots,t_T$, we suppose that the demonstrator provides several trajectories, each consisting of a sequence of states and inputs $(\mathbf{x},\mathbf{u})$, which we assume to be near-optimal,

similar to previous work for demonstration learning [29]. Given that the joint estimation of $c_{t_i}(\cdot)$ and $\mathcal{X}_{t_i}$ is difficult and often ambiguous, we resort to state-of-the-art techniques to carry out cost and constraint estimation independently. Hence, for each sub-task example demonstrations, t_i, we collect D demonstrations where the optimal trajectory is not influenced by avoiding the unsafe region $\bar{\mathcal{X}}_t$, i.e., the complement of $\mathcal{X}_t$, as well as D demonstrations where the optimal trajectory is influenced by avoiding the unsafe region $\bar{\mathcal{X}}_t$. While the former demonstrations are indicated with $(\hat{\mathbf{x}}, \hat{\mathbf{u}})_{t_i(d)}$, where $t_i(d)$ with $d \in [1, \dots, D]$ represents the d^{th} trajectory sample for task t_i and the hat notation indicates sub-optimality, the later demonstrations are denoted as $(\hat{\mathbf{x}}, \hat{\mathbf{u}})^s_{t_i(d)}$, respectively.

2) Learning from demonstrations:

a) Learning of cost function and parametric mapping: This step jointly estimates the cost parameter m in expression (2) and the *parametric mapping* $\mathcal{M}$, both to be used in the online pipeline. It is comprised of two sub-steps, the *principal component extraction* of embedding vectors and the *learning of the unknown cost and mapping parameters*, which are detailed in the following.

Principal component extraction: The embedding vector representations for the descriptions of the T sub-task examples, satisfying Assumption 1, are computed using a transformer architecture [33, 34]. These embeddings, denoted as $e_1, \dots, e_T$, typically reside in a high-dimensional space ($\mathbb{R}^s$, where $s \geq 300$). However, since our focus is on the dimensions that exhibit significant variation across sub-tasks, rather than capturing the full semantic content of each sentence, we can often represent them in a much lower-dimensional space. Therefore, we compress the embeddings, obtaining $\tilde{e}_1, \dots, \tilde{e}_T$ and hence, reduce the number of demonstrations required to learn the parametric mapping in the subsequent step.

Learning of the unknown parameters: Once the principal component embedding vectors $\tilde{e}_1, \dots, \tilde{e}_T$ are computed, the joint estimation problem leverages the demonstrations $(\hat{\mathbf{x}}, \hat{\mathbf{u}})_{t(d)}$ and tools from IRL to jointly learn the parametric mapping function $\mathcal{M}$ that maps from the compressed embedding vector $\tilde{e}_{\bar{t}}$ to the task specifying feature selection vector $\theta_{\bar{t}}$ whose structure is pre-determined and parametrized via the parameter vector q, i.e., $\mathcal{M}: \; q, \tilde{e}_{\bar{t}} \mapsto \theta_{\bar{t}}$, as well as the cost parameter m, shared among all tasks in $\mathcal{S}$.

b) Learning of constraints: Provided a cost function $c(\cdot)_t$ has been learnt for example sub-task t, it is possible to use the demonstrations $(\hat{\mathbf{x}}, \hat{\mathbf{u}})^s_{t(d)}$, avoiding an unsafe region $\bar{\mathcal{X}}_t$, to learn its over-approximation and consequently estimate the parameter vector ρ of a parameterized constraint description $g((x, u), \rho) \leq 0$ as in expression (2) that under-approximates the safe region $\mathcal{X}_t$.

B. Online Robotic Control

In what follows we briefly describe the components of the proposed online pipeline.

1) Language Module:

a) Task Planner (TP): This block receives from the user a natural language command to perform a complex task $\mathcal{P}$. Using a pre-trained LLM, it outputs a list of sub-tasks $\tau_1, \tau_2, \dots$ to be executed sequentially to fulfill the command.

b) Sub-task Validation: This block ensures that the sub-tasks output by the task planner are sufficiently similar to the example sub-tasks which have been provided by the demonstrator. This comparison is performed by embedding each entry in the list of sub-tasks, and verifying that these embeddings are similar to the embeddings of the example sub-tasks. Let e_{sub} denote the full-dimensional embedding of a sub-task output by the task planner, and let $e_1, \dots, e_T$ denote full-dimensional embeddings of the demonstration sub-tasks. Then the sub-task validation test verifies that for each sub-task,

$$\left\| \begin{bmatrix} e_1 & e_2 \dots & e_T \end{bmatrix}^\dagger e_{\mathsf{sub}} \right\| \leq v, \tag{3}$$

where v is a user-defined threshold. This check is motivated by characterizations from multi-task learning [35] which measure task relatedness via the coverage coefficient between the target task and the space spanned by the source tasks.[1]

c) Optimization Designer (OD): This block sequentially receives each of the sub-task embedding vectors e_{τ_i}, passing the sub-task validation, and passes it through the precomputed *parametric mapping* to obtain θ_{τ_i}.

2) Control Module:

a) Trajectory Generator (TG): This block receives the parameter vector θ_{τ_i} computed by the OD. It solves for an optimal trajectory via an MPC scheme (1), with cost and constraint sets structured as in expression (2) and parameters m and ρ extracted from the offline computation.

b) Trajectory Tracker (TT): This block receives the trajectory generated by the TG, and maps it to low-level actions for the given hardware, i.e., torque inputs applied to the motors of a robot, etc. Further details are provided in [11].

V. Experiments

The proposed pipeline has been extensively tested in simulation, and its effectiveness has been demonstrated on a real robotic setup. The chosen environment, as well as the obtained results, are detailed in the following subsections.

A. Environment

The efficacy of DEMONSTRATE is evaluated using the custom simulation environment of [11], integrated into PandaGym [36]. It consists of a seven-axis robotic arm (Franka Panda) on a table. We consider three environments, denoted with "Cubes", "Sponge", and "Drawer". In "Cubes", there are four randomly placed cubes of different colors, i.e., *green*, *blue*, *red*, and *orange*, in "Sponge", there is a pan placed on a table with a sponge next to it, and in "Drawer", there is a cabinet with a drawer set on the table. "Sponge" and "Drawer" are used in simulation only, while the "Cubes" environment

[1] In the multi-task literature, the relatedness would be measured by the distance of the weights on the shared basis $\theta_{\bar{t}}$. However, in our setting these weights are output by a neural network mapping the language embeddings to the weights. As this network is trained on the demonstrated sub-tasks, outputs from any embedding tend to be mapped to weights which are similar to the weights for the demonstrated sub-tasks.

is used for both, simulation and hardware experiments. An illustration of the environments is given in Figure 3. The "Drawer" environment has not been present in the baselines' simulation environment but was added to test the sub-task validation, as well as explore and demonstrate the limits of generalization. It is hence not part of the baseline comparison.

B. Experiment Design

In this section, we detail the target tasks, designed to test our newly proposed pipeline.

1) Target tasks of online pipeline: In all four environments, we execute *pick and place movements* with different objects, belonging to a common task space $\mathcal{S}$. The state of the object and the robot are known throughout the experiment. In the "Cubes" environment, we conduct three different experiments. Task one is described as "stack all cubes" and is considered successfully solved if all four cubes are stacked on top of each other, task two as "build a pyramid with two cubes at the base and one at the top", and the third task follows as "write the letter L flat on the table". In the "Sponge" environment the robot arm is tasked to "grab the sponge and clean the plate with circular movements". Finally, in the "Drawer" environment, the robot is asked to "open the drawer x cm". Each of these tasks is composed of simple sub-tasks that involve moving the gripper some distance and bearing relative to the objects in the scene. The language description for an example of one such sub-task is "0.05 meters right of object one."

2) Components of offline pipeline: We use the pipeline with embedding vectors e_t obtained from the 'all-mpnet-base-v2' sentence transformer model [34]. The parametric mapping $\mathcal{M}(q, e_t)$ is chosen as a multi-layer perceptron regressor with four hidden layers of 512 neurons and ReLu activation functions. The embedding e_t is projected onto 20 principal components formed by the sub-task examples. It is trained using 20 demonstrations for each of the 90 different sub-tasks. Since the number of required demonstrations increases with increasing suboptimality, we train the framework in simulation, using a demonstrating controller, which completes the required sub-tasks examples. The obtained results are intended to serve as a proof of concept, for practical settings in which real-world demonstrations would be used.

C. Benchmarking Results

The performance of DEMONSTRATE, is evaluated for each task, with a total of 50 runs conducted. The results are then compared against those obtained using the existing NARRATE, VoxPoser [37], and Code as Policies [7] architecture. These baseline evaluation results are from [11]. In order to highlight the efficacy of the proposed methodology, we determine the *Success Rate (SR)* and identify the underlying causes of failure. These are categorized as follows: *Task Planner Failure (TP)*, *Optimization Designer Failure (OD)*, and *Collision (CO)*. A task planner failure occurs when the sub-tasks output by the task planner do not lead to execution of the user command. An *Optimization Designer Failure* occurs if the run was not successful due to an inaccurate encoding of the sub-task into the optimization problem, meaning that the control objective is not sufficiently accurate to achieve the desired task. In the baselines, optimization designer failures occur due to coding errors resulting in code that is not executable. In DEMONSTRATE, these failures arise due to insufficient precision of the learned cost and constraint functions, such as the placement of blocks with insufficient precision on top of each other, resulting in an unstable and collapsing stack. Finally, *Collision* is indicated when the gripper or an object that the gripper is holding collides with another object, resulting in a failed execution of the task. The obtained percentages are presented in the Table I. It can be seen, that DEMONSTRATE shows comparable or higher success rates than the best performing base line model in all 4 tasks.

D. Sub-task Validation

While the Trajectory Generator (TG) outputs an OCP that aligns with the designed task space $\mathcal{S}$, there are still occasions where the TP suggests sub-tasks that are not closely related in their task description with the available sub-task examples. The sub-task validation procedure of (3) is critical to catching these instances. In Appendix Section B, we provide illustrative examples where the sub-task validation procedure does and does not interfere. For the proposed DEMONSTRATE architecture in Figure 1, failure of sub-task validation results in re-planning by the TP, up to a maximum number of tries (set to 5 for the experiments of Table I). If the planner fails to generate a list of valid sub-tasks within these trials, the system refuses to attempt the task. In the experiments of Table I, the task planner always achieves a valid list of sub-tasks within three attempts. Re-planning was triggered twice for "Stack" and "Pyramid", and not at all for "L" and "Wipe Pan".

E. Limits of Generalization

Table II shows the results of the procedure applied to the task of opening a drawer, where the demonstration data consisted of sub-tasks moving between 4 and 12 centimeters to the side of an object. The results on the drawer demonstrate that a user command asking the arm to open a drawer to distances close to those demonstrated results in correct movement. Asking for larger distances results in the sub-task validation detecting the task to be "not similar enough" to the available sub-task examples and refusing to perform it.

F. Hardware

The applicability of the proposed approach is demonstrated through real-world experimentation, utilising a Franka Emika Panda robotic arm and two external realsense cameras, as illustrated in Figure 3. The effectiveness of the method is evaluated on a subset of the previously introduced tasks, specifically "Stack", "L", and "Pyramid", relying on point cloud estimation to extract the poses of the cubes. A sequence of executed steps for the tasks "Stack" and "Pyramid" are visualized in Figure 2.

TABLE I: Simulation Experiments: Comparison of DEMONSTRATE (pre-trained embedding model) against NARRATE and VoxPoser on three tasks, 50 repetitions per experiment. We report the *Success Rate (SR)* for each method and the failure reason categorized into *Task Planner Failure (TP)*, *Optimization Designer Failure (OD)*, and *Collision (CO)*

Method	Stack		L-Shape		Pyramid		Wipe Pan	
	SR [%]	TP/OD/CO [%]	SR [%]	TP/OD/CO [%]	SR [%]	TP/OD/CO [%]	SR [%]	TP/OD/CO [%]
CaP	**98**	2/0/0	10	16/14/60	16	76/6/2	22	48/20/10
VoxPoser	26	0/26/48	0	0/24/76	16	22/24/38	48	22/24/6
NARRATE	92	6/0/2	58	30/0/12	76	0/2/22	62	34/0/4
DEMONSTRATE	88	6/6/0	**60**	30/2/8	**80**	10/0/10	**94**	2/0/4

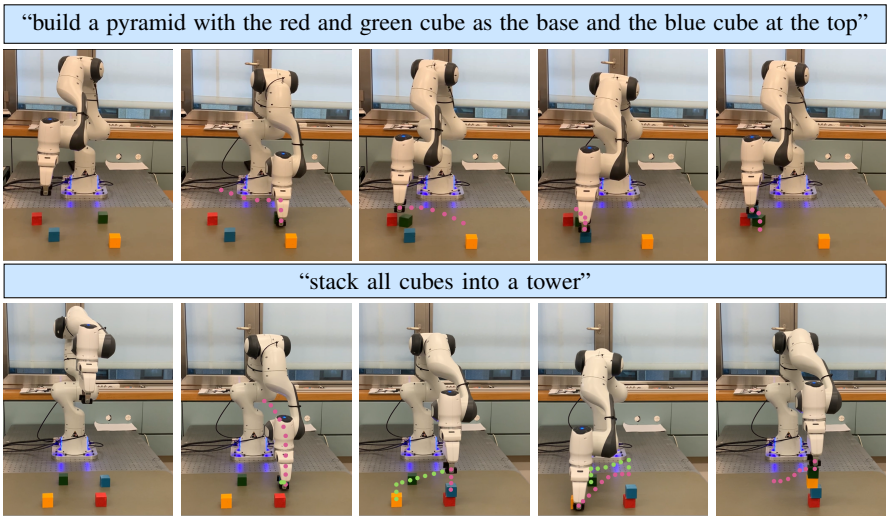

Fig. 2: Visualization of hardware execution to build a pyramid (top) and stack all cubes (bottom). End-effector trajectories are indicated with dotted lines, whereas pink trajectories are executed before green trajectories.

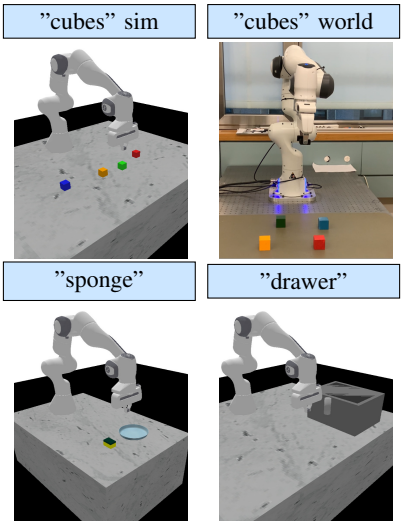

Fig. 3: Illustration of considered environments in simulation and real-world experiments.

“Open the Drawer ”	+“5cm”	+“10cm”	+“15cm”
Result (cm)	5.8	8.9	Cannot perform

TABLE II: We task the robot to “Open the drawer ” finished by either “5cm”, “10cm”, or “15cm”, and show the resulting final position for the drawer.

VI. Conclusion

Contributions: We developed a new methodology that makes use of tools derived from inverse reinforcement learning and multitask learning in order to address the challenges associated with prompt design, prompt sensitivity and LLM hallucinations in LLM-based controller design. This is achieved by (i) utilising demonstrations instead of in-context examples consisting of mathematical expressions and (ii) using entropy maximization based inverse reinforcement learning to establish a direct mapping from embeddings of natural language to weights defining the corresponding objective function. This avoids the use of the LLM for complex objective generations, and instead only leverages it to establish sub-tasks which can be composed to execute the user's command.

Results: The results in tabletop manipulation with a robot arm represent a proof of concept. In particular, they demonstrate that by obtaining a collection of expert demonstrations for a variety of sub-tasks, one can use language-conditioned inverse reinforcement learning with a high-level task planner to execute a diverse array of tasks. The method achieves success rates similar to or better than previous language in controlling approaches on a number of benchmark tasks in Section V-C; however, it does so at the expense of poorer zero-shot generalization to tasks far from the collected demonstrations, shown in Section V-E. To prevent generalization failures, we introduce *Sub-task* validation, tested in Section V-D, which prevents the execution of tasks too different from those used for training.

Limitations: Building upon state-of-the-art methods from inverse reinforcement learning and constraint learning, the framework currently relies on individual demonstration collection in constrained and unconstrained environments. This is cumbersome and will be addressed in future improvements, e.g., by including the constraints in the objective function of the OCP. Additionally, it was demonstrated in Section V-E that the framework does not excel in zero-shot generalization to user commands whose sub-tasks are very different from those demonstrated. This is part of a tradeoff inherent in replacing the LLM used for optimization design with language conditioned inverse reinforcement learning. Improving generalization with the proposed framework would require scaling up the expert demonstrations for a broader range of sub-tasks. To move beyond the proof of concept example to a more practically useful framework, one could separate numerical information from semantic information in the sub-task description, and process them separately. This may improve generalization behavior of the type in Section V-E. Finally, the framework was tested only in a proof of concept example where it was feasible to quickly collect demonstrations in simulation in a controlled manner. An interesting avenue for future work would be to apply the framework to more sophisticated langauge-enabled robotic control examples, e.g. by using existing datasets of tasks performed by human demonstrators [38].

References

[1] Y.-J. Wang, B. Zhang, J. Chen, and K. Sreenath, "Prompt a robot to walk with large language models," *arXiv:2309.09969*, 2023.

[2] Y. Jiang, A. Gupta *et al.*, "Vima: General robot manipulation with multimodal prompts," *arXiv:2210.03094*, 2022.

[3] M. Shridhar, L. Manuelli, and D. Fox, "Perceiver-actor: A multi-task transformer for robotic manipulation," in *Conf. on Robot Learning*, 2022.

[4] Z. Mandi, S. Jain, and S. Song, "Roco: Dialectic multi-robot collaboration with large language models," *arXiv:2307.04738*, 2023.

[5] J. Austin, A. Odena *et al.*, "Program synthesis with large language models," *arXiv:2108.07732*, 2021.

[6] D. Trivedi, J. Zhang *et al.*, "Learning to synthesize programs as interpretable and generalizable policies," *Adv. Neural Inf. Process. Syst.*, vol. 34, pp. 25 146–25 163, 2021.

[7] J. Liang, W. Huang *et al.*, "Code as policies: Language model programs for embodied control," in *IEEE Int. Conf. on Robot. and Automat. (ICRA)*. IEEE, 2023, pp. 9493–9500.

[8] P. Goyal, S. Niekum, and R. J. Mooney, "Using natural language for reward shaping in reinforcement learning," *arXiv:1903.02020*, 2019.

[9] W. Yu, N. Gileadi *et al.*, "Language to rewards for robotic skill synthesis," *arXiv:2306.08647*, 2023.

[10] Y. J. Ma, W. Liang *et al.*, "Eureka: Human-level reward design via coding large language models," *arXiv:2310.12931*, 2023.

[11] S. Ismail, A. Arbues *et al.*, "Narrate: Versatile language architecture for optimal control in robotics," in *Int. Conf. Intel. Robots Syst. (IROS)*. IEEE, 2024, pp. 9628–9635.

[12] S. Arora and P. Doshi, "A survey of inverse reinforcement learning," *Artif. Intell.*, 2021.

[13] A. Maurer, M. Pontil, and B. Romera-Paredes, "The benefit of multitask representation learning," *J. Mach. Learn. Res.*, vol. 17, no. 81, pp. 1–32, 2016.

[14] Y. Zhang and Q. Yang, "A survey on multi-task learning," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 12, pp. 5586–5609, 2022.

[15] Q. K. Luu, X. Deng *et al.*, "Context-aware llm-based safe control against latent risks," *arXiv:2403.11863*, 2024.

[16] O. Y. Lee, A. Xie *et al.*, "Affordance-guided reinforcement learning via visual prompting," *arXiv:2407.10341*, 2024.

[17] T. Brown, B. Mann *et al.*, "Language models are few-shot learners," *Adv. Neural Inf. Proc. Syst.*, vol. 33, pp. 1877–1901, 2020.

[18] H. Zhang, H. Song *et al.*, "A survey of controllable text generation using transformer-based pre-trained language models," *ACM Computing Surveys*, vol. 56, no. 3, pp. 1–37, 2023.

[19] Z. Jiang, F. F. Xu *et al.*, "How can we know what language models know?" *Trans. Assoc. Comput. Linguistics*, vol. 8, pp. 423–438, 2020.

[20] T. Shin, Y. Razeghi *et al.*, "Autoprompt: Eliciting knowledge from language models with automatically generated prompts," *arXiv:2010.15980*, 2020.

[21] A. Zou, Z. Wang *et al.*, "Universal and transferable adversarial attacks on aligned language models," *arXiv:2307.15043*, 2023.

[22] M. Deng, J. Wang *et al.*, "Rlprompt: Optimizing discrete text prompts with reinforcement learning," *arXiv:2205.12548*, 2022.

[23] T. Zhang, X. Wang *et al.*, "Tempera: Test-time prompting via reinforcement learning," *arXiv:2211.11890*, 2022.

[24] L. Reynolds and K. McDonell, "Prompt programming for large language models: Beyond the few-shot paradigm," in *Extended abstracts of the 2021 CHI conference on human factors in computing systems*, 2021, pp. 1–7.

[25] J. Park, S. Lim *et al.*, "Clara: Classifying and disambiguating user commands for reliable interactive robotic agents," *IEEE Robot. Automat. Lett.*, vol. 9, no. 2, pp. 1059–1066, 2024.

[26] A. Z. Ren, A. Dixit, A. Bodrova, S. Singh, S. Tu, N. Brown, P. Xu, L. Takayama, F. Xia, J. Varley, Z. Xu, D. Sadigh, A. Zeng, and A. Majumdar, "Robots that ask for help: Uncertainty alignment for large language model planners," 2023.

[27] B. Kouvaritakis and M. Cannon, "Model predictive control," *Switzerland: Springer Int. Publishing*, 2016.

[28] B. D. Ziebart, A. L. Maas *et al.*, "Maximum entropy inverse reinforcement learning," in *Conf. Artif. Intel.*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[29] S. Levine and V. Koltun, "Continuous inverse optimal control with locally optimal examples," in *Proc. Int. Conf. Mach. Learn.*, 2012, pp. 475–482.

[30] G. Chou, D. Berenson, and N. Ozay, "Learning constraints from demonstrations," in *Algorithmic Found. of Robot. XIII*. Springer, 2020, pp. 228–245.

[31] G. Chou, N. Ozay, and D. Berenson, "Learning parametric constraints in high dimensions from demonstrations," in *Conf. Robot Learn.* PMLR, 2020, pp. 1211–1230.

[32] M. Lázaro-Gredilla, J. Quinonero-Candela *et al.*, "Sparse spectrum gaussian process regression," *J. Mach. Learn. Res.*, vol. 11, pp. 1865–1881, 2010.

[33] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proc Conf. Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[34] K. Song, X. Tan *et al.*, "Mpnet: Masked and permuted pre-training for language understanding," *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 16 857–16 867, 2020.

[35] T. T. Zhang, B. D. Lee *et al.*, "Guarantees for nonlinear representation learning: non-identical covariates, dependent data, fewer samples," *arXiv:2410.11227*, 2024.

[36] Q. Gallouédec, N. Cazin *et al.*, "panda-gym: Open-source goal-conditioned environments for robotic learning," *arXiv:2106.13687*, 2021.

[37] W. Huang, C. Wang *et al.*, "Voxposer: Composable

3d value maps for robotic manipulation with language models," *arXiv:2307.05973*, 2023.

[38] A. O'Neill, A. Rehman, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain *et al.*, "Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0," in *Int. Conf. Robot. Automation (ICRA)*. IEEE, 2024, pp. 6892–6903.

[39] Y. Shoukry, P. Nuzzo, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada, "Smc: Satisfiability modulo convex optimization," in *Proceedings of the 20th international conference on hybrid systems: Computation and control*, 2017, pp. 19–28.

APPENDIX A
DETAILS FOR OFFLINE COMPUTATIONS

For the blocks of the offline pipeline, we present solution approaches that are either derived from or build upon existing results in the corresponding domain. Note, however, that depending on the intended application and future developments in the considered field, the presented methods may be adapted or replaced.

A. Principle Component Analysis of Embedding Vectors

To achieve the compression of the embedding vectors, we apply Principal Component Analysis (PCA) as follows. We stack the embedding vectors of our example tasks into a matrix $E = [e_1^\top, \dots, e_T^\top]^\top \in \mathbb{R}^{T\times s}$ and center the data around their column means, resulting in $\tilde{E} = E - \mathbf{1}_{s,1}(\frac{1}{T}\sum_{t=1}^{T} e_t)$, where $\mathbf{1}_{s,1}$ denotes a vector of length s filled with ones. We then compute the singular value decomposition $\tilde{E} = U\Sigma V^\top$. Finally, ordering singular values and their corresponding rows and columns of U and $V^\top$ in a descending order, a reduction to z principal components is achieved by $\tilde{E}_{\mathrm{PC}} = U_z\Sigma_z$.

B. Inverse Reinforcement Learning for Parameter Estimation

To jointly learn the parameter vector q of the parametric mapping function $\mathcal{M} : \ q, \tilde{e}_t \mapsto \theta_t$, as well as the cost parameter m, we propose a formulation that builds upon the entropy maximization approach in [29], which allows for unknown parameter estimation in a parametrized reward functions $c((\mathbf{x},\mathbf{u});w)$ from D suboptimal demonstrations, indicated with $(\hat{\mathbf{x}},\hat{\mathbf{u}})_{(d)}$ for $d = 1,\dots,D$. The estimation follows from the fact that the probability of obtaining $(\hat{\mathbf{u}})_{(d)}$ for the given $(\hat{x}_0)_{(d)}$ and parameter w is

$$P((\hat{\mathbf{u}})_{(d)}|(\hat{x}_0)_{(d)};w) = \frac{1}{Z}\exp\left(c((\mathbf{x},\mathbf{u});w)\right) \tag{4}$$

where we denote with Z the normalization term $Z = \int_{\mathbf{u}} \exp\left(c((\mathbf{x},\mathbf{u});w)\right) d\mathbf{u}$. Resorting to the concept of maximum entropy [28], the parameters w are defined to maximize the logarithm of likelihood (4), which, as shown in [29], can be approximated as

$$\begin{aligned}&\log(P((\hat{\mathbf{u}})_{(d)}|(\hat{x}_0)_{(d)};w))\\ &\approx \frac{1}{2}g_{(d)}^\top H_{(d)}^{-1} g_{(d)} + \frac{1}{2}\log|-H_{(d)}| - \frac{d_u}{2}\log(2\pi),\end{aligned}$$

with $g = \nabla_u c((\hat{\mathbf{x}},\hat{\mathbf{u}})_{(d)};w)$ and $H_i = \nabla_u^2 c((\hat{\mathbf{x}},\hat{\mathbf{u}})_{(d)};w)$. Hence, the parameter estimate over all collected demonstrations follows as

$$\hat{w} = \arg\max_w \sum_{d=1}^{D} \frac{1}{2}\left(g_{(d)}\right)^\top H_{(d)}^{-1} g_{(d)} + \frac{1}{2}\log|-H_{(d)}|.$$

We extend this setup to the multi-task setting for a collection of tasks $t = 1,\dots,T$ and sum the likelihood over all available tasks in our task-space $\mathcal{S}$. We further leverage the structure in (2) and replace the parameter vector w with the parametric

mapping $\mathcal{M}(q, e_t) \rightarrow \theta_t$ as well as the cost parameter m. This results in the following optimization problem:

$$\hat{q}, \hat{m} = \arg\max_{q,m} \sum_{t=1}^{T} \sum_{d=1}^{D} \log P((\hat{\mathbf{u}})_{(d)} | (\hat{x}_0)_{(d)}; q, e_t, m),$$

which consequently, learning for a cost instead of a reward function, can be approximated as

$$q, \hat{m} = \tag{5}$$
$$\arg\min_{q,m} \sum_{d=1}^{D} \sum_{t=1}^{T} -\frac{1}{2} \left(g_{t(d)}\right)^{\top} H_{t(d)}^{-1} g_{t(d)} - \frac{1}{2} \log |-H_{t(d)}|,$$

where $H_{t(d)} := \nabla^2_{\mathbf{u}} \sum_k \mathcal{M}(q, \tilde{e}_t)^{\top} \phi((x_k, u_k)_{t(d)}; m)$ and $g_{t(d)} := \nabla_{\mathbf{u}} \sum_k \mathcal{M}(q, \tilde{e}_t)^{\top} \phi((x_k, u_k)_{t(d)}; m)$.

C. Constraint Learning from Safe Demonstrations

Following the approach in [31], we simulate inputs at random and keep those $(\hat{\mathbf{x}}, \hat{\mathbf{u}})^{us}_{t(d)}$ that satisfy $c_t((\hat{\mathbf{x}}, \hat{\mathbf{u}})^{us}_{t(d)}) < c_t((\hat{\mathbf{x}}, \hat{\mathbf{u}})^{s}_{t(d)})$, and hence violate the constraints (as otherwise they would have been attainable by the optimal solution). Obtaining A unsafe simulated trajectories, denoted as $(\hat{\mathbf{x}}, \hat{\mathbf{u}})^{us}_{t(d,a)}$ for $a = 1, \ldots, A$, the parameter ρ is the solution to

$$\text{find } \rho \tag{6a}$$
$$\text{s.t. } g((\hat{x}_k, \hat{u}_k)^{s}_{t(d)}, \rho) \leq 0, \ \forall k = 1, \ldots, N-1 \tag{6b}$$
$$g((\hat{x}_k, \hat{u}_k)^{us}_{t(d,a)}, \rho) > 0, \ \exists k = 1, \ldots, N-1, \tag{6c}$$
$$\forall t = 1, \ldots, T, \ \forall d = 1, \ldots, D, \ \forall a = 1, \ldots, A. \tag{6d}$$

Estimating an unsafe area $\bar{\mathcal{X}}_t$ that can be described as a boolean conjunction of general convex inequalities, satisfiability modulo convex optimization (SMC) can be employed [39]. For this purpose, the problem in equation (6) is reformulated as follows:

$$\begin{aligned}
&\text{find } \rho, \mathbf{b}^{us}, \mathbf{b}^{s} \\
&\text{s.t. } b^{s}_{(d,a,k)} \rightarrow g((\hat{x}_k, \hat{u}_k)^{s}_{(d)}, \rho) \leq 0 \\
&\quad \sum_{k=0}^{N-1} b^{s}_{d,a,k} = N \\
&\quad b^{us}_{(d,a,k)} \rightarrow g((\hat{x}_k, \hat{u}_k)^{us}_{(d,a)}, \rho) > 0 \\
&\quad \sum_{k=0}^{N-1} b^{us}_{d,a,k} \leq N - 1 \\
&\quad \forall k = 0, \ldots, N-1, \ \forall d = 1, \ldots, D, \ \forall a = 1, \ldots, A.
\end{aligned}$$

Where $\mathbf{b}^{us} = (\mathbf{b}^{us}_{(1,1)}, \ldots, \mathbf{b}^{us}_{(D,A)}) \in \mathbb{R}^{NAD}$ and $\mathbf{b}^{s} = (\mathbf{b}^{s}_{(1,1)}, \ldots, \mathbf{b}^{s}_{(D,A)}) \in \mathbb{R}^{NAD}$ define concatenations of the boolean vectors $\mathbf{b}^{us}_{(d,a)} = (b^{us}_{(d,a,0)}, \ldots, b^{us}_{(d,a,N-1)}) \in \mathbb{R}^{N}$ and $\mathbf{b}^{s}_{(d,a)} = (b^{s}_{(d,a,0)}, \ldots, b^{s}_{(d,a,N-1)}) \in \mathbb{R}^{N}$, which allow for the logic assignment of each convex constraint to an individual binary variable. Such a variable is then evaluated as one, for a fulfilled constraint and zero for a violated constraint, making it possible to indicate constraint violation via their summation and finally find a combination of ρ, $\mathbf{b}^{us}$, and $\mathbf{b}^{s}$ that fulfills all the constraints.

APPENDIX B
DETAILS FOR SUB-TASK VALIDATION

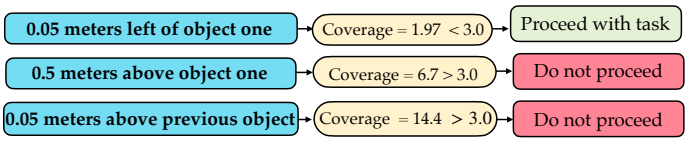

Fig. 4: Examples of sub-task validation using the coverage coefficient computed in (3) with threshold $t = 3.0$.

In Figure 4, we provide illustrative examples where the sub-task validation procedure does and does not interfere, setting the threshold v in (3) to 3.0. In the first instance, the sub-task is of the form provided by the demonstrations, containing a distance along with a maneuver relative to objects one to four. In this case, the sub-task distance is below the threshold. In the second case, a distance considerably larger than those demonstrated is provided, and the third case references the previous sub-task, which is not conform with the provided annotations, both leading to a large coverage coefficient. We provide the coverage coefficients for each instance in an attempt to demonstrate the sensitivity of our sub-task validation operator to the threshold value v. We show that doubling v would result in the rejection of the same instances as with the current threshold value.