Asking the Right Questions: Facilitating Semantic Constraint Specification for Robot Skill Learning and Repair

Aaquib Tabrez*, Jack Kawell*, and Bradley Hayes

***Abstract*— Developments in human-robot teaming have given rise to significant interest in training methods that enable collaborative agents to safely and successfully execute tasks alongside human teammates. While effective, many existing methods are brittle to changes in the environment and do not account for the preferences of human collaborators. This ineffectiveness is typically due to the complexity of deployment environments and the unique personal preferences of human teammates. These complications lead to behavior that can cause task failure or user discomfort. In this work, we introduce *Plan Augmentation and Repair through SEmantic Constraints* (PARSEC): a novel algorithm that utilizes a semantic hierarchy to enable novice users to quickly and effectively select constraints using natural language that correct faulty behavior or adapt skills to their preferences. We show through a case study that our algorithm efficiently finds corrective constraints that match the user's intent, providing a path for novice users to exploit the advantages of constrained motion planning combined with human-in-the-loop skill training.**

I. Introduction

The increased availability and prevalence of collaborative robotics has led to growth in our expectations for human-robot teaming and accordingly to the roles and responsibilities assigned to autonomous systems. Robots that collaborate or work in close proximity with humans have safety-critical requirements imposed on their autonomy, conditioned on task-specific and collaborator-specific parameters. As these deployments become increasingly widespread, their complexity and impact of failure will grow in kind. Consequently, a desirable and pertinent trait for such collaborative agents is the ability to accommodate human users' preferences [12] and requirements [22]. For example, an assistive robot designed to work in elder care environments should take into consideration the different comfort levels of individuals with whom the robot works (e.g. a desired minimum distance). Without such considerations, the robot might potentially cause physical or emotional harm should it behave in a manner that violates expectations [6].

Furthermore, it is highly unlikely that these types of interactions will only occur in the exact environments in which such robots were trained, increasing the likelihood of unexpected or dangerous behavior. This generalization is a central tenet of intelligent automation: being able to utilize a model trained in one environment within a different one [24]. The cost functions being used by these systems to plan or otherwise compute their behavior may not account for crucial factors such as novel environmental artifacts and user requirements or preferences. In this work, we introduce an interactive algorithm to help those who use these systems to add constraints into a robot's planner to create safer, more robust skills that better accommodate user specifications.

The authors are all affiliated with the Department of Computer Science at the University of Colorado Boulder. {mohd.tabrez, jack.kawell, bradley.hayes}@colorado.edu.

* These authors contributed equally to this work

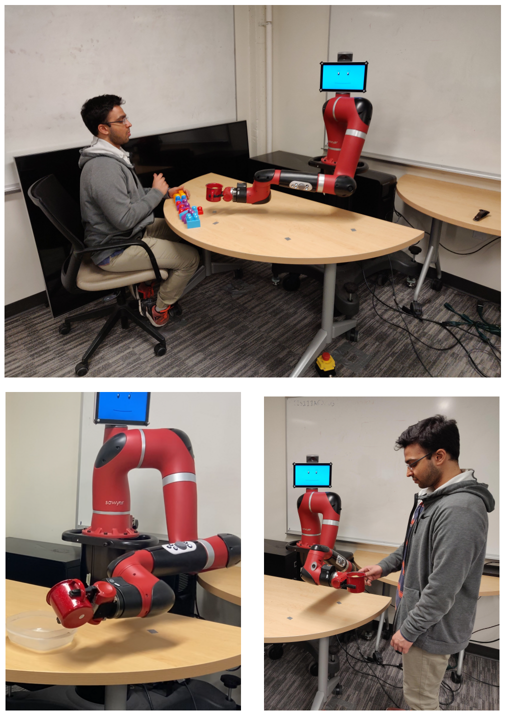

Fig. 1: User interacting with a Sawyer robot. Three tasks are shown (top to bottom, left to right): 1) A cleaning task where Sawyer attempts to move the cup from one side of the table to the other in front of the user; 2) A pouring task where Sawyer attempts to pour the contents of the cup into another container; 3) A handover task where Sawyer attempts to hand the cup to the user without spilling the contents.

It is clear that robots must be able to adapt their behaviors to changes in their environment, as well as to the personal preferences of humans they encounter, to be successful without also levying a burden on those around them. Thus despite the many challenges it poses, in-situ learning will be essential as even modern robots require experts to reprogram them or guide them in the retraining of a skill [22]. Even with state-of-the-art learning from demonstration techniques, retraining skills to achieve reliable performance and predictable behavior takes considerable time and effort [2] or expertise [18]. In order for robots to be able to adapt their skills to novel environments and shifting user preferences, we posit that new techniques enabling non-experts to leverage the power of constrained motion planning are required.

In response to this technical challenge, we present *Plan Augmentation and Repair through SEmantic Constraints* (PARSEC): a novel algorithm that utilizes a semantic hierarchy to enable novice users to use natural language to quickly select and parameterize constraints that can be applied within a constrained motion planner to correct faulty behavior or adapt skills to accommodate preferences. Core to this methodology is the intuition that novice users must be able to interact in a natural way with the robot and that constraint discovery is greatly accelerated by organizing constraints as leaves in a semantic tree of parameterizations. Our method uses plain language explanations given by a user to bootstrap a brief iterative query process that leads to the specification of an allowable constraint set that matches their intent. The intuitiveness of this process enables skill correction by those without robotics or motion planning experience, making it suitable for a wide audience. The two primary contributions of our work are:

- PARSEC, a human-in-the-loop algorithm that facilitates constraint annotation for motion planning problems via a novel hierarchical semantic process
- An experimental validation and evaluation of PARSEC, assessing its performance in three different robotic case studies using human feedback and demonstrating a statistically significant time reduction for skill correction compared to baseline.

II. Background and Related Work

Learning from Human-in-the-loop. Much work has been done analyzing the ability of human feedback to improve robot skill performance. St. Clair and Matarić showed the effectiveness of robot verbal feedback in human-robot task collaborations [21]. Additionally, Sadigh et al. presented an approach for robot production of social communication during human-robot task collaboration to improve in situ decision-making and team performance [19] and Meriçli et al. contributed a method which utilizes corrective human demonstration as a complement to an existing hand-coded algorithm for improving task performance [16].

Similar works look into cognitive inspired architectures that help infer task constraints from natural language and demonstrate through user studies that natural language is the preferred instructions method for modifying robot skills [20], [27]. Our proposed method infers the most likely correction of the problem, and then initiates communication with the user to resolve the ambiguity before the skill is augmented.

Learning from Demonstration. Researchers have also worked on learning from failed demonstrations. In a paper by Grollman et al., humans are assumed to be sub-optimal and incapable of performing a task correctly. Their failed demonstrations are then used as negative constraints on the robot's exploration [10]. The same group of researchers in other work speculated that in higher dimensions, additional information from the user will most likely be necessary to enable efficient failure-based learning [11]. Our proposed system applies this type of information from the user to improve interaction efficiency during failure correction.

Other researchers have focused on learning robot objective functions from human guidance through physical corrections provided by the person while the robot is acting [2]. A key limitation of this technique is that it requires users with a technical background to perform the skill correction which keeps novice users from being able to benefit from it [28]. Instead of physical demonstrations, humans typically use speech to provide high-level goals or teleoperation commands for autonomy [9]. Kramer et al. [14] compared four natural language understanding models, evaluating their performance to understand domestic service robot commands by recognizing the actions and any complementary information in them. These models learn possible correspondences between parsed instructions and candidate groundings that include objects, regions and motion constraints. In the realm of learning from demonstration (LfD) there has been much focus on repairing faulty skills or even training new skills with only a single demonstration and then providing fine tuned skill adjustment through a user interface [15], [18].

Learning through user preference and querying. Researchers have also worked on learning user preferences over trajectories taken by robotic manipulators. Abdo et al. used a collaborative filtering model to learn user preferences about how best to organize objects in their environment [1]. However, Bobu et al. showed how assuming that a human's desired objective lies within the robot's hypothesis space can lead to irrelevant task corrections [5]. These works demonstrate the importance of having a feedback loop between the user and the robot so that correction can occur without confusion.

Querying users for improving performance and learning has been an active field of research as well [26]. Cakmak et al. categorized types of queries users preferred based on the informativeness and ease of answering [7]. Another approach has been for enabling a robot to recover from failures by generating targeted assistance requests [23]. Similarly, Biyik et al. showed another approach of learning through queries focused on generating easy questions through greedy maximization of information gain [4]. In Volosyak et al., the system actively queries the human for task goals or execution assistance, and through speech the user provides a high-level (e.g. "pour a drink") and low level (e.g. "gripper up") instruction [25]. To the best of our knowledge, we believe this is the first work that combines learning from human feedback, constrained motion planning, and in-situ iterative querying of a human user (using natural language) to augment and repair robot skills.

III. Methods

In this section, we introduce **PARSEC** (Plan Augmentation and Repair through SEmantic Constraints), an interactive method whereby a robot iteratively queries a human collaborator to determine how to apply constraints to its motion planner for improving its skill performance or robustness. Our approach enables non-expert users to correct faulty robot skills through natural language feedback, which our

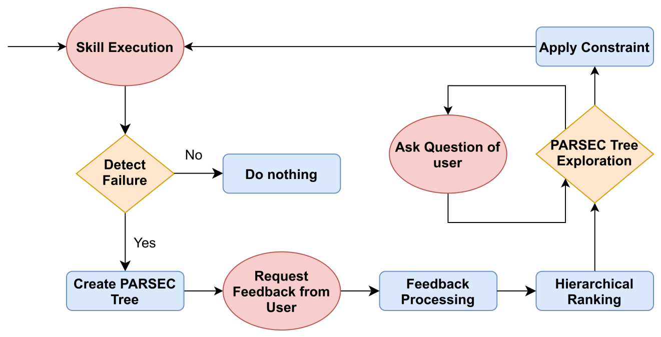

Fig. 2: Execution loop of PARSEC, beginning with skill execution and proceeding through constraint specification.

algorithm processes and maps to parameterized constraints (e.g., 'stay at least 15cm away from the human').

While constrained motion planning has been shown to be a powerful tool in improving robot task and skill performance, the selection and parameterization of which constraints to apply remains an open, time consuming problem [13], [15]. Our insight is that if we can structure the available constraints and their parameterizations to *maximize the information utility of each question* and algorithmically reduce the problem space *based on the user's feedback*, we can substantially reduce the level of effort required to incorporate constrained motion planning into learning from demonstration and human-in-the-loop skill repair.

Preliminaries. PARSEC is a post-hoc method applicable once the learning agent has been trained to execute a specific task within a training environment, meaning the agent already has the foundational elements of the planning problem defined (i.e., motion planner, goal states, and cost function). The outcome of PARSEC is a list of parameterized constraints to apply to the motion planning problem, with the intent that the application of these constraints will prevent failure modes not initially captured by the planner's cost function. One intuitive use case is that constraints can be used to fill in for 'common sense' (or user preferences) that the cost function may not properly encode, such as applying the constraint that a cup in the gripper must always be upright, since the planner's cost function may not encode avoiding spilling the cup's contents.

We define a **constraint** to be a Boolean function mapping a state of the world to *true* if that state is not in violation of the constraint represented within the function and *false* otherwise (akin to STRIPS predicates [8]). For example, min_distance(object_1, object_2, distance_in_cm): $State \rightarrow \{True, False\}$ could be a constraint that evaluates to *true* only when object_1 is at least distance_in_cm from object_2 in the provided state vector (e.g., *min_distance(cup, laptop, 10)* would return *true* if the cup and laptop are at least 10 cm apart, *false* otherwise). Within PARSEC, we characterize the parameters for constraints as either belonging to a discrete finite set (e.g., "objects": ['parts bin', 'table', 'block'] - a list of object names in the environment) or representing a continuous or innumerable set (e.g. "distance": a real-valued quantity expressed in centimeters). These sets represent the domain knowledge of the learning agent that it can use for specifying and communicating about constraints.

We use this domain knowledge to create an informed structural framework for our query mechanism to efficiently solicit user feedback, resulting in a more rapid constraint specification process that requires less human effort. In this section, we detail our method of intelligently querying the human collaborator for determining beneficial motion planning constraints.

A. PARSEC Algorithm

Being a post-hoc method, PARSEC is intended to be applied after initial skill specification or learning. The execution flow of PARSEC can be seen in Fig. 2. We model skill execution as solving a motion planning problem from some initial state (s_0) to any one of a set of goal states ($s_g \in \bar{G}$). More specifically, our work is intended to be applied within domains modelled as Markov Decision Processes defined by (S,A,T) where **S** is the set of states, **A** is a set of actions the agent can choose from, and **T** $: S \times A \times S \rightarrow \mathbb{R}$ is the transition function that provides the likelihood of transitioning between two states given an action.

PARSEC requires a Boolean signal to indicate whether the robot has successfully completed its task or if it has failed (Fig 2: Detect Failure step). In collaborative Human-Robot Interaction scenarios, failure may be indicated by adverse human behaviors (e.g., human retreating from the robot/workspace, showing annoyance, etc,.) resulting from execution of the skill (as opposed to not being able to plan from start to goal state). The entry point for our approach is Algorithm 1, which interactively produces a list of parameterized motion planning constraints when given a motion planning problem specification, planner, and set of possible constraint functions.

PARSEC Walkthrough. In lines 1-2, we attempt to create a successful plan for the problem as-specified using the initial state s_0, set of goal states $\bar{G}$, and known constraints K. If a viable plan is generated, PARSEC returns K as no additional constraints are necessary for completion. Otherwise, the algorithm begins the iterative, interactive process of constraint discovery. In line 3, PARSEC creates a tree (Fig 3) of available constraint functions and their potential parameter assignments (described later in Algorithm 3). This tree forms the basis for our query mechanism and leverages its inherent structural benefits for effective search. In line 6, $RequestExplanation()$ solicits the user for open-ended semantic feedback to describe how the skill is failing.

Line 7 refines the PARSEC Tree based on this feedback, scoring each node's estimated relevance to optimize the order in which they are used to form queries. To build intuition for how the PARSEC Tree is refined, consider the example in Figure 3, where a robot may be executing a pouring skill in the vicinity of a human by picking up a cup, moving it over a target area, then pouring out its contents. Given the feedback "The cup was too close", the [cup, distance] parameterization node in the middle of the tree would become the first node queried in line 10, skipping other nodes that might normally precede it ([objects], [cup, john], etc.). In lines 9-12, the robot asks if the node is relevant to the constraint they wish to add to the planner, iterating through the ordered list of the root's children until a relevant node is found (eventually returning with no solution if no node is identified by the human). The $AskQuestion(node)$ call in line 10 initiates a recursive exploration down the tree until a leaf node is confirmed (indicating a fully parameterized constraint to add), described in Algorithm 2. Finally, lines 14-15 are responsible for adding the newly identified constraint and testing the skill to see if the new formulation is successful.

Inspired by the success of Bajcsy et al.'s method of incremental skill repair [2], our method is architected to focus on providing one new constraint with each iteration of the main while loop until a successful skill is produced.

Algorithm 1: Plan Augmentation and Repair through Semantic Constraints (PARSEC)

```
Input:  Motion Planner $Planner$, Start state $s_0$,
        Goal state set $\bar{G}$, Known Constraints $K$, List
        of constraint functions $\bar{C}$, List of parameter
        types $\bar{S}$, Dictionary mapping parameter types
        to lists of valid assignments $P$
Output: List of parameterized constraints for skill
        segment $s_0 \rightarrow \bar{G}$ or $False$ on failure
1  constraints ← $K$;
2  if Planner.plan($s_0, \bar{G}$, constraints) $\neq \emptyset$ then return;
3  tree ← CreatePARSECTree($\bar{C}, \bar{S}, P$);
4  while do
5      // No successful plan from $s_0$ to a state in $\bar{G}$
6      response ← RequestExplanation();
7      rankedTree ← ScoreTree(tree, response, $\bar{S}, P$);
8      found ← False; new_constraints ← $\emptyset$;
9      for node $\in$ rankedTree do
10         new_constraint ← AskQuestion(node);
11         if new_constraint != $\emptyset$ then
12             found←True; break;
13     if found is False then return False;
14     constraints.append(new_constraint);
15     if Planner.plan($s_0, \bar{G}$, constraints) $\neq \emptyset$ then
           break;
16 return constraints;
```

B. PARSEC Tree Creation

The process for creating the PARSEC Tree (Fig. 3), used as the basis for query production, is described in Algorithm 3. The PARSEC Tree is a data structure with fully parameterized constraints at its leaves, meant to efficiently guide a user through the process of selecting a constraint function and values for its parameters. To interactively select a parameterized constraint, one could perform a depth-first search, asking if the contents of the current node are relevant: descending one level if 'yes', and moving laterally if 'no'. In PARSEC, we utilize semantic feedback from a human to reduce the number of nodes and levels of the tree, further reducing the level of effort required to select parameterized constraints.

Algorithm 2: AskQuestion

```
Input: PARSEC Tree Node $node$
Output: Fully Parameterized Constraint Node
1  // $node$ consists of a constraint function from $\bar{C}$ and
     has parameters of types contained in $\bar{S}$ from Alg. 1
2  // Ask user if the parameters indicated by this node
     are correct
3  response ← AskUser(node.parameters);
4  if response is "No" then return False; //Wrong node;
5  if node.children is $\emptyset$ then return node;
6  // $node$ is relevant but not a leaf node: search deeper
7  for child in node.children do
8      ans ← AskQuestion($child$);
9      if ans is "No" then continue;
10     else return ans;
```

The nodes of the PARSEC tree contain either parameter types (e.g., 'objects', 'humans', 'robots', 'distance', etc.), combinations of parameter values (e.g., 'cup', 'table', ['cup','table'] etc.), or parameterized constraint functions (e.g., above('cup','table')). In Algorithm 3, three inputs are required: a set of constraint functions ($\bar{C}$), a set of parameter types ($\bar{S}$), and a parameter value dictionary P mapping types (elements of $\bar{S}$) to a list of valid values for each.

The set $\bar{C}$ consists of all constraint function signatures available to the robot (e.g., ($\bar{C} = \{$ above(object,object), below(object,object), min_distance(object,human,distance)$\}$). $\bar{S}$ consists of all discrete parameter *types*, and is used to help logically cluster the parameter values within the tree. Finally, P provides all of the possible parameter values that the human could choose from, and is used to form the majority of internal tree nodes. To accommodate continuous valued constraint parameters within P, we add them as if they were each a single discrete parameter value (e.g., 'distance') and lazily ground them to specific values at the end of the constraint selection process.

Following Algorithm 3, nodes are organized such that each parent node is a subset of its children, where additional detail is added at each level of the tree until a fully parameterized constraint is reached at the leaves. We organize the tree to resolve parameter values down to constraint functions under the intuition that there will generally be many more possible constraint functions (which are robot-centric) than valid parameter types (which are environment-centric) within a given scenario.

While the PARSEC Tree is helpful in guiding users to specify valid constraints, the simple '20 Questions'-style depth-first search procedure described above is tedious and can be improved by utilizing the human for more than a simple Boolean signal.

C. Feedback Processing

During PARSEC, the robot asks the user to explain how the skill should be corrected. This open-ended explanation is

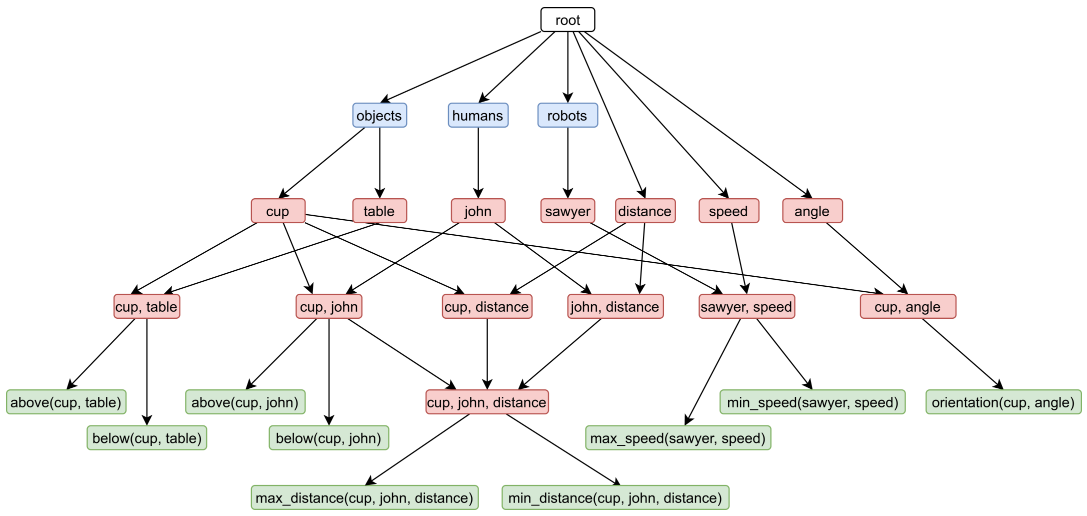

Fig. 3: Example of a partial PARSEC tree. Blue nodes represent parameter types, red nodes represent both grounded (e.g., 'cup') and lifted (e.g., 'angle') parameters and combinations of parameters, and green nodes represent fully parameterized constraints. Lifted parameters are resolved to grounded values after they are assigned to a parameterized constraint (III-B).

parsed by a natural language processing (NLP) engine that scores each tree node in relation to the feedback given. In a practical sense, if the user's explanation refers to specific objects or attributes in the environment they will receive a higher score than objects that were not. For example, if the robot's domain knowledge consists of the words "computer" and "person" and the user gives the explanation "Don't move near the computer" the scoring process will give a high value to "computer" and a low value to "person". For our implementation, we used the Python Natural Language Toolkit (NLTK) [3] with the WordNet [17] lexical database.

Since $\bar{S}$ is the set of all parameter types and P contains all discrete parameter values and continuous parameter identifiers, then the total working dictionary for the robot is D where $D = \bar{S} \cup P$. Each element $d_i \in D$ is assigned a set of exact match words E_i and a similarity match words N_i. Processing a user explanation works by iterating over each word w_j in the explanation and assigning a value v_i to the each element $d_i \in D$ according to the formula:

$$v_i = \begin{cases} 1 & w_j \in E_i \\ sim(w_j, N_i) & w_j \notin E_i \end{cases}$$

Where $sim(w_j, N_i)$ is a function that returns the highest similarity score of the word w_j when compared to each word in N_i.

D. Node Relevance Scoring

The PARSEC algorithm can then utilize the scores given to each element of its working dictionary to score each node within the PARSEC tree. This is done using the scoring function:

$$score = \sum_{i=1}^{|\bar{V}|} v_i + \frac{\prod_{i=1}^{|\bar{V}|} v_i}{|\bar{V}|}$$

Where $|\bar{V}|$ is the number of parameters that the node encapsulates and $v_i \in \bar{V}$ is the value given to each parameter by the feedback processing step. The intuition behind this function is that the summation has more value if a node consists of parameters with more positive scores. The product component will also be larger when the node's parameters contain non-zero scores, but it is discounted by the number of parameters to keep the tree iteration from diving too deep into the tree without high confidence in the parameters of the node. This scoring function is used by Algorithm 1 (Line 7) as *ScoreTree* to rank each node in the tree.

Additionally, if the above scoring function returns a value of 0 and $|\bar{V}| = 1$ (the node has only a single parameter), the score is set to a small positive value ϵ. This ensures that nodes near the top of the tree will be prioritized if no information is known (all other nodes have $score = 0$).

IV. Evaluation and Results

We evaluated PARSEC using human feedback to provide corrective constraint annotations for a Rethink Robotics Sawyer robot (Figure 1) as it executed a collection of three representative manipulation planning tasks. As our algorithm is meant to expedite the constraint annotation process, the primary objective metric of this evaluation is the number of questions required to identify a constraint (and its parameterization) that allows the skill's motion planner to successfully plan and execute the desired behavior. This metric was chosen as a proxy for measuring the amount of time and effort expended by the user to correct faulty behavior from the robotic agent.

The three evaluation tasks are:

- *Handoff Task:* A handoff task where the robot attempts to give a cup to a user but spills the contents in the process of moving the cup towards the user. This scenario is an example of faulty training where the skill needs correction to repair a poor training. A constraint that keeps the cup upright will repair this skill.
- *Pouring Task:* A pouring task where the robot is tasked with pouring the contents of a cup into a receptacle

Algorithm 3: CreatePARSECTree

```
Input: Set of potential constraint functions $\bar{C}$, Set of
       constraint function parameter types $\bar{S}$,
       Dictionary mapping parameter types to lists
       of valid assignments $P$
Output: Tree of parameter types, values, and
        parameterized constraints (Fig. 3)
1  max_args ← max(num_function_args(c) for c in $\bar{C}$);
2  min_args ← min(num_function_args(c) for c in $\bar{C}$);
3  tree ← Graph(); root ← Node('root');
4  tree.add_vertex(root);
5  for $c$ in $\bar{C}$ do
6      if num_function_args(c) $\neq 0$ then continue;
7      tree.add_vertex(Node(c));
8      tree.add_edge(root, Node(c));
9  prev_level ← [];
10 for $s$ in $\bar{S}$ do
11     tree.add_vertex(Node(s));
12     tree.add_edge(Node(root),Node(s));
13     prev_level.append(Node(s))
14 for $i$ in range(1,max_args+1) do
15     cur_level ← [];
16     param_combinations = List of all
         parameterizations (using $P$) of $i$-length elements
         from power set of arg lists in $\bar{C}$;
17     for $p_{combo}$ in param_combinations do
18         tree.add_vertex(Node($p_{combo}$));
19         $cur\_level$.append(Node($p_{combo}$));
20         for $p_{node}$ in prev_level do
21             if $p_{node} \subset p_{combo}$ or $p_{combo}$ in
                 $P[p_{node}.name]$ then
                 tree.add_edge($p_{node}$, $cur\_level$[-1]) ;
22         for $c$ in $\bar{C}$ do
23             if num_function_args(c) $\neq i$ then continue;
24             // Add function $c$ parameterized by $p_{combo}$
25             tree.add_vertex(Node(c($p_{combo}$)));
26             tree.add_edge(Node($p_{combo}$),Node(c($p_{combo}$)))
27     prev_level ← cur_level;
28 return tree;
```

sitting on a table. Though the skill has been properly trained, in this scenario the receptacle has been moved from its position during training and so the robot pours the contents in the wrong position over the table. This demonstrates a situation where the skill has been correctly trained but is not robust to changes in the environment. A constraint that requires the cup to be above the receptacle during the pouring motion will correctly augment this skill.

- *Cleaning Task:* A cleaning task where the robot moves a cup across the surface of a table in front of a user who is performing another task sitting at the table. The robot moves too closely to the user causing discomfort to the person. This scenario is an example of training that doesn't correctly represent the user's preferences for how the robot should act. A constraint that keeps the robot at a comfortable distance specified by the user will correct this skill.

A. Case Study Setup

We trained a Rethink Robotics Sawyer robot arm to perform the three three tasks described in IV. For each task, we trained the robot using Concept-Constrained Learning from Demonstration [18] both to successfully reach each skill's goal state as well as to fail in some aspect of the execution as described above (by removing an important learned constraint). We then recorded videos of the robot succeeding and failing in the skill execution so that we had examples of the faulty skill and correct skill in each example.

We then created a survey that requested the participant to view the videos of each skill and describe in a single sentence how the skill ought to be corrected. The participants could view the videos as often as they wished and had access to the videos of both the faulty skill execution as well as the correct skill. This way they could directly compare the faulty execution to the corrected behavior and so judge what specifically needed to be changed. The survey asked them to describe the correction as if they were talking to the robot.

We received 19 total responses to the survey with one response having to be discarded due to the participant not following the directions properly. Participants had varying levels of experience with robots, ranging from novice to expert. These 18 responses were used as the user feedback input for each algorithm.

We tested three approaches for the skill correction that utilized the explanations provided by users in the survey to correct the three faulty skills:

1) **NLP Method**: Rank the available parameterized constraints (leaves of PARSEC tree) for the planner based on the scores returned by the scoring function then querying the user for the correct constraints by iterating through the sorted list.
2) **Tree Method**: Run the PARSEC algorithm (Alg. 1) while omitting the tree scoring from line 7, setting $rankedTree$ equal to $tree$ from line 3. This explores the PARSEC Tree from the root node using Algorithm 2 but with no prioritization of the traversal ordering by the user's semantic feedback.
3) **Tree-NLP Method**: Use the full Plan Augmentation and Repair through SEmantic Constraint (PARSEC) algorithm (Algorithm 1), leveraging the user's semantic feedback to accelerate traversal through the tree.

Our experiment set out to investigate the following hypotheses: **H1:** The number of queries in the PARSEC condition will be lower compared to the Semantic baseline (NLP method) and Naïve Exploration of PARSEC Tree method (Tree method), and **H2:** Naïve Exploration of PARSEC Tree and PARSEC will perform better in comparison to NLP method (due to the structural benefit from the PARSEC Tree).

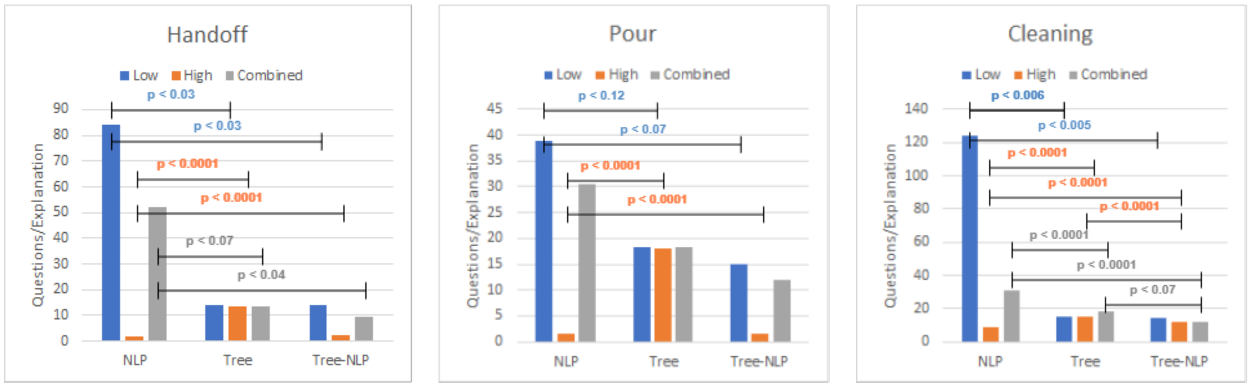

Fig. 4: Results for all tasks, with user-provided explanations binned into low-quality, high-quality, and combined categories.

B. *Results*

We analyze each algorithm's performance on each task with the user feedback from our survey, calculating the number of questions asked before the correct constraint was discovered. Due to the large variety of potential PARSEC Tree constructions and the effect that node ordering would have on the results, each algorithm was run for 100 trials with the ordering of child nodes shuffled (before the ranking step) to account for any 'lucky' ordering effect of nodes with equal scores. This meant that that for each task we compiled a 100×18 table of data for each of the three tasks (18 from survey responses), where rows represent algorithm effectiveness per explanation and the columns represent the number of algorithm runs given that explanation. These tables can be averaged across the rows or columns to analyze different aspects of the results:

- Averaging across rows gives information about how each algorithm performs on the skill as a whole given the user feedback data. We call this type of averaging: *average by skill*.
- Averaging across columns gives information about how each algorithm performs on each explanation given by the users. This is informational in diagnosing high quality versus low quality explanations. We call this type of averaging: *average by explanation*.

For *average by explanation*, we did not observe any multimodalities in the distributed data but from the response we noticed that some of the users were descriptive about the recommendation (high quality explanation) and some users were vague about the failure and how should robot correct it (low quality explanation). This led to higher variance in the number of queries for repair (especially in the NLP condition). Therefore, to get the better insight we segmented our user provided explanations into three bins: 1) high quality explanations, 2) low quality explanations, and 3) combined (all the explanation). We conducted an ANOVA to test effects across our three algorithmic approaches for the average number of queries for each task based on the quality of explanation. For the handoff task, we found a significant effect from the PARSEC algorithm on number of queries for combined explanations $(F(2,51) = 3.91, p < 0.03)$, **confirming H1**. Post-hoc comparisons using Tukey's HSD test (Figure 4) revealed that using PARSEC tree resulted in a significant different level of queries, **confirming H2**. Similarly, we can see that both high quality explanations $(F(2,18) = 145.83, p < 0.0001)$ and low quality explanation $(F(2,30) = 5.39, p < 0.01)$ **validate H1 and H2**.

The significant effect was observed for the cleaning task using the combined explanations $(F(2,51) = 7.3, p < 0.002)$ and post-hoc comparison show similar results as the handoff task (i.e., using PARSEC tree resulted significant different level). Similarly, observation can be seen effectiveness of PARSEC tree from low quality explanations $(F(2,24) = 15.8, p < 0.0001)$. The high quality explanations show statistical significance $(F(2,24) = 152.8, p < 0.0001)$ for number of queries and Tukey's HSD test gives significant different level for each of the approaches. No significant effects were found with respect to combined explanation for the pouring task measure of number of queries (p = 0.1) but we found the significant effect of the our algorithm for high quality explanations $(F(2,9) = 10379, p < 0.0001)$ and Post-hoc analysis reveal statistical significant different levels because of the PARSEC tree. Results for low quality explanation $(F(2,39) = 3.12, p < 0.06)$ are inconclusive (Figure 4) but merit further investigation to confirm an effect.

Likewise, for *average by skill* results, we conducted an ANOVA to investigate differences between our three algorithmic approaches for the handoff task, cleaning task, and pouring task. For the handoff task, significant effects were found from the usage of PARSEC algorithm on number of queries $(F(2,297) = 3362.48, p < 0.0001)$ and Tukey's HSD test (Figure 5) reveals a significantly different level of queries for each approach. Outcomes were similar for the cleaning $(F(2,297)$ = 2135.47, p $<$ 0.0001) and pouring tasks $(F(2,297)$ = 454.24, p $<$ 0.0001) further **validating H1 and H2**.

V. Conclusions

In this work, we present *Plan Augmentation and Repair through SEmantic Constraints* (PARSEC), a new algorithm that enables novice robot users to quickly correct faulty behavior or apply personal preferences to a robot skill through a process informed by a NLP accelerated semantic hierarchy of queries. Our results show that PARSEC reduces the number of queries the user is required to answer before

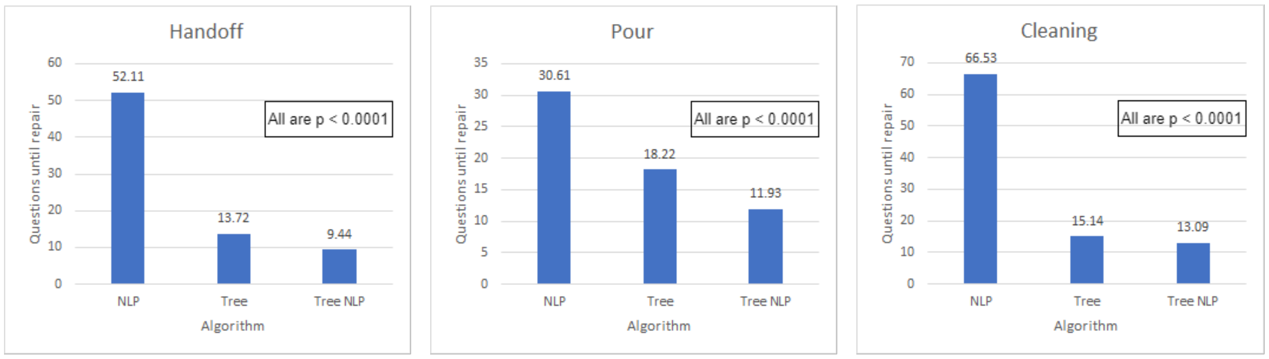

Fig. 5: Results for all three evaluation domains, showing average performance by each method. The full PARSEC algorithm (Tree-NLP) provides a consistently more efficient experience in each task.

the skill is corrected as compared to a baseline algorithm only applying semantic rankings to constraints and a baseline algorithm that utilizes hierarchical structure to direct queries for resolving desired parameterized constraints. In demonstrating the benefit of combining the PARSEC Tree's hierarchical structure alongside a semantic analysis of the user's feedback, we contribute a novel method for human-in-the-loop skill learning that merges human-robot interaction and constrained motion planning.

Our primary result shows that PARSEC reduces time spent by users across three representative manipulation tasks, each demonstrating a different application domain: correcting a skill with faulty or incomplete training (handover task), augmenting a skill to perform in a novel environment (pouring task), and adapting a skill to user preferences (cleaning task). These results show the ability for PARSEC to be applied to various types of robotic skills while at the same time providing an efficient way for novice users to correct and adapt the robot's behavior to their own preferences.

VI. Acknowledgements

Special thanks to Himanshu Gupta and Liam Merz Hoffmeister for their help with Python implementation and NLP development. This work was supported in part by NSF NRI Award #1830686 and ARL STRONG Cycle 2.

References

[1] N. Abdo, C. Stachniss, L. Spinello, and W. Burgard. Robot, organize my shelves! tidying up objects by predicting user preferences. In *2015 IEEE international conference on robotics and automation (ICRA)*.

[2] A. Bajcsy, D. P. Losey, M. K. O'Malley, and A. D. Dragan. Learning from physical human corrections, one feature at a time. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, pages 141–149, 2018.

[3] E. L. Bird, Steven and E. Klein. *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.

[4] E. Bıyık, M. Palan, N. C. Landolfi, D. P. Losey, and D. Sadigh. Asking easy questions: A user-friendly approach to active reward learning. *arXiv preprint arXiv:1910.04365*, 2019.

[5] A. Bobu, A. Bajcsy, J. F. Fisac, and A. D. Dragan. Learning under misspecified objective spaces. *arXiv preprint arXiv:1810.05157*, 2018.

[6] J. Broekens, M. Heerink, H. Rosendal, et al. Assistive social robots in elderly care: a review. *Gerontechnology*, 8(2):94–103, 2009.

[7] M. Cakmak and A. L. Thomaz. Designing robot learners that ask good questions. In *2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 17–24. IEEE, 2012.

[8] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*.

[9] T. Fong, C. Thorpe, and C. Baur. *Collaborative control: A robot-centric model for vehicle teleoperation*, volume 1. Carnegie Mellon University, The Robotics Institute Pittsburgh, 2001.

[10] D. H. Grollman and A. Billard. Donut as i do: Learning from failed demonstrations. In *In International Conference on Robotics and Automation*, 2011.

[11] D. H. Grollman and A. G. Billard. Robot learning from failed demonstrations. *International Journal of Social Robotics*, 2012.

[12] B. Hayes and B. Scassellati. Challenges in shared-environment human-robot collaboration. In *"Collaborative Manipulation" Workshop at the 8th ACM/IEEE International Conference on Human-Robot Interaction.*, page 8, 2013.

[13] L. Jaillet and J. M. Porta. Path planning under kinematic constraints by rapidly exploring manifolds. *IEEE Transactions on Robotics*, 2012.

[14] E. R. Kramer, A. O. Sáinz, A. Mitrevski, and P. G. Plöger. Tell your robot what to do: Evaluation of natural language models for robot command processing. In *Robot World Cup*. Springer, 2019.

[15] M. B. Luebbers, C. Brooks, C. L. Mueller, D. Szafir, and B. Hayes. Arc-lfd: Using augmented reality for interactive long-term robot skill maintenance via constrained learning from demonstration.

[16] Ç. Meriçli, M. Veloso, and H. L. Akın. Task refinement for autonomous robots using complementary corrective human feedback. *International Journal of Advanced Robotic Systems*, 8(2):16, 2011.

[17] G. A. Miller. Wordnet: A lexical database for english. 1995.

[18] C. Mueller, J. Venicx, and B. Hayes. Robust robot learning from demonstration and skill repair using conceptual constraints. 2018.

[19] D. Sadigh, A. D. Dragan, S. Sastry, and S. A. Seshia. Active preference-based learning of reward functions. In *Robotics: Science and Systems*, 2017.

[20] M. Scheutz, E. Krause, B. Oosterveld, T. Frasca, and R. Platt. Spoken instruction-based one-shot object and action learning in a cognitive robotic architecture. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, 2017.

[21] A. St. Clair and M. Mataric. How robot verbal feedback can improve team performance in human-robot task collaborations. In *Proceedings of the tenth annual acm/ieee international conference on human-robot interaction*, pages 213–220, 2015.

[22] A. Tapus, M. Maja, and B. Scassellatti. The grand challenges in socially assistive robotics. *IEEE Robotics and Automation Magazine*, 14(1):N–A, 2007.

[23] S. Tellex, R. Knepper, A. Li, D. Rus, and N. Roy. Asking for help using inverse semantics. 2014.

[24] S. Thrun and L. Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[25] I. Volosyak, O. Ivlev, and A. Graser. Rehabilitation robot friend ii-the general concept and current implementation. In *9th International Conference on Rehabilitation Robotics*. IEEE, 2005.

[26] N. Wilde, D. Kulić, and S. L. Smith. Learning user preferences in robot motion planning through interaction. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018.

[27] T. Williams, G. Briggs, B. Oosterveld, and M. Scheutz. Going beyond literal command-based instructions: Extending robotic natural language interaction capabilities. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[28] S. Wrede, C. Emmerich, R. Grünberg, A. Nordmann, A. Swadzba, and J. Steil. A user study on kinesthetic teaching of redundant robots in task and configuration space. *Journal of Human-Robot Interaction*.