Evaluating Methods for End-User Creation of Robot Task Plans

Chris Paxton,[1] Felix Jonathan,[1] Andrew Hundt,[1] Bilge Mutlu,[2] and Gregory D. Hager[1]

***Abstract*— How can we enable users to create effective, perception-driven task plans for collaborative robots? We conducted a 35-person user study with the Behavior Tree-based CoSTAR system to determine which strategies for end user creation of generalizable robot task plans are most usable and effective. CoSTAR allows domain experts to author complex, perceptually grounded task plans for collaborative robots. As a part of CoSTAR's wide range of capabilities, it allows users to specify SmartMoves: abstract goals such as "pick up component A from the right side of the table." Users were asked to perform pick-and-place assembly tasks with either SmartMoves or one of three simpler baseline versions of CoSTAR. Overall, participants found CoSTAR to be highly usable, with an average System Usability Scale score of 73.4 out of 100. SmartMove also helped users perform tasks faster and more effectively; all SmartMove users completed the first two tasks, while not all users completed the tasks using the other strategies. SmartMove users showed better performance for incorporating perception across all three tasks.**

I. Introduction

The relatively recent development of human-safe robots has spurred a new wave of innovation in commercial, end-user-programmable systems such as the Rethink Robotics Sawyer, Universal Robots UR5, and Franka Emika robots. Robotic systems in the laboratory are becoming ever more intelligent, flexible, and robust, and these advances are increasingly translated into applications in industry. As a result, today's manufacturers seek skilled workers with strong problem-solving skills and a STEM background as a part of the transformation to "Industry 4.0," focused on smart and interconnected facilities [1]. The workers and factories of the future require intelligent, powerful tools that allow them to utilize the best capabilities of modern robots. However, whether or not even skilled workers and expert users can translate the complex perception, planning, and control capabilities offered by modern collaborative robots into industrial efficiency is unknown.

These developments have led to a growing interest in making it easy for domain experts to transfer knowledge to collaborative robots, either through a user interface [2], [3], [4], [5], natural language [6], or learning from demonstration [7], [8], [9], [10]. To take full advantage of these systems, the human user must have an accurate mental model of a robot's capabilities [11]. Therein lies a potential problem, as people tend to think of robot motions abstractly, e.g., "grab the next available component for this assembly," instead of in terms of basic robotics concepts like fixed spatial positions.

Fig. 1: Users interacted with the CoSTAR system in different ways. Left: a user positions the robot to teach it a new waypoint. Right: a user fine tunes part of a tree using the BT-based user interface.

We previously proposed the CoSTAR system as a potential solution. [3] CoSTAR offers a set of planning and perceptual capabilities, united through a Behavior Tree-based task plan editor, that is designed to enable expert users to author complex plans for collaborative robots. We also demonstrated that CoSTAR allows users to solve a wide variety of problems such as sanding, assembly, and wire bending [5]. It provides users a great deal of flexibility by offering many different ways to specify actions.

These new "SmartMoves" were designed mimic the abstract way people think about manipulation tasks. To perform a `SmartGrasp` or a `SmartRelease` operation, one of the two new actions we describe in this paper, a user demonstrates a grasp or a position. Then they specify the associated object types and abstract descriptions like "left of

[1]Department of Computer Science, Johns Hopkins University, 3400 N Charles Street, Baltimore, MD 21218, USA, {cpaxton, fjonath1, ahundt1, hager}@jhu.edu

[2]Department of Computer Sciences, University of Wisconsin–Madison, 1210 W Dayton Street, Madison, WI 53706, USA bilge@cs.wisc.edu

[3]Source code for the CoSTAR system is available on GitHub: https://github.com/cpaxton/costar_stack

the robot" that they want for that operation.

In this paper, we examine the usability of CoSTAR's Behavior Tree-based user interface, and explore different strategies for using perception to create grounded task plans. To represent different strategies for incorporating advanced capabilities such as perception and motion planning, we compare three versions of the CoSTAR system without high-level abstraction against CoSTAR with SmartMoves:

(1) *Simple*: a blind version of the system with only the ability to servo to pre-programmed waypoints in joint space.
(2) *Motion Planning*: a system that uses perception with motion planning to avoid obstacles.
(3) *Relative Motion*: a system that can detect object positions but requires that users explicitly specify how to interact with each object.

By contrast, (4) *SmartMove* integrates perception and planning via abstract queries for objects matching a specified predicate. This condition allows users to more easily specify pick-and-place tasks via the addition of the new high-level `SmartGrasp` and `SmartRelease` operations. Users found CoSTAR and Behavior Trees to be a usable and even enjoyable system; the higher levels of abstraction present in condition (4) allowed for improved task performance and better generalization to new environments. [4] In short, in this work we contribute:

- Two new high-level actions to the existing CoSTAR framework, `SmartGrasp` and `SmartRelease`, designed to improve usability and generalization.
- Comparative analysis of different strategies for grounding end-user task plans via machine perception.
- Usability analysis of CoSTAR and each of its variants.

II. Background

Recent approaches for end-user instruction of collaborative robots include the development of new user interfaces [2], [3], [4], learning from demonstration [7], [10], or systems that make use of natural language together with ontologies and large knowledge bases to follow high-level instructions, such as Tell Me Dave [6] or RoboSherlock [12].

Our proposed user interface is based on Behavior Trees, which have previously been used on humanoid and surgical robots, among other applications [13], [14], [15]. Others have explored designing user interfaces for robot task specification [2], [4], [5]. Nguyen et al. [2] describe ROS Commander as a user interface based on finite state machines for authoring task plans. Similarly, Steinmetz and Weitschat [4] describe a graphical tool called RAFCON.

Previous work in robot task specification has shown complex [12], [16] or interactive behavior [17] without examining the specification of this behavior. Dantam et al. [17] specify a complex motion grammar that allows a human to interactively play chess with a robot, where the robot must pick up and manipulate every piece on the board. Methods for Task and Motion Planning allow planners to integrate selection of task parameterizations with computation of motion plans that will satisfy its requirements [16], [18]; these methods are an inspiration for incorporating SmartMove into CoSTAR.

An alternate approach to direct task specification is to learn tasks from expert demonstrations. Alizadeh et al. [7] learn skills which can be re-used according to a PDDL planner. Levine et al. [8] proposed reinforcement learning methods for effectively learning individual skills with a demonstration as a prior. In these cases, the end user still needs a way to connect individual skills. Dianov et al. [9] take a hybrid approach, using task graph learning to infer task structure from demonstrations and a detailed ontology. Other recent work explored combining learned actions with sampling-based motion planning and a high-level task specification [10].

III. The CoSTAR System

CoSTAR is a Behavior Tree-based user interface that aims to facilitate user interaction through a combination of an intuitive user interface, robust perception, and integrated planning and reasoning operations. It is designed to be reliable, capable and cross-platform and was applied to both the KUKA LBR iiwa and Universal Robots UR5 robot. It is implemented as a component-based framework, where each component exposes a set of distinct operations that can be composed as a task plan [5].

The underlying task is represented as a Behavior Tree (BT). BTs allow us to visually construct complex, concurrent behaviors out of the equivalent of programming constructs such as IF-statements, TRY-CATCH blocks, and FOR- and WHILE-loops. In this section, we describe the basics of our BT implementation and the CoSTAR framework. For more information on Behavior Trees, see prior work [19], [5].

A. Overview of Behavior Trees

The basic operation of the BT is the "tick:" a status check sent at some high frequency (e.g., 60 hz) and propagated through the tree according to the rules associated with each node. Every node has an associated action that is performed when it is ticked; if a node is ticked, it will return some value in $\{\texttt{SUCCESS}, \texttt{FAILURE}, \texttt{RUNNING}\}$. The **Root** of a BT has a single child node, generally a logical node, and is the source of all ticks.

Complex task structure is achieved via **Logical** and decorator nodes. These control program flow and the order of operations. CoSTAR's logical nodes include:

- Sequence (`->`): Tick all children in order. This will stop when a tick reaches a running child.
- Selector (`?`): Tick all children in order until one returns success. This is used to construct IF-statements with multiple cases.
- Parallel-All (`|A|`): Tick all children in parallel; return success if all children return success and failure otherwise.
- Repeat: Tick all children some (possibly infinite) number of times

[4] Supplementary video of experiments and of the CoSTAR system in use is available on YouTube, including expert instruction of the CoSTAR system and highlights from trials: https://www.youtube.com/watch?v=TPXcWU-5qfM&list=PLF86ez-NVmyFMuj10dkUkgGlGpcM5Vok9

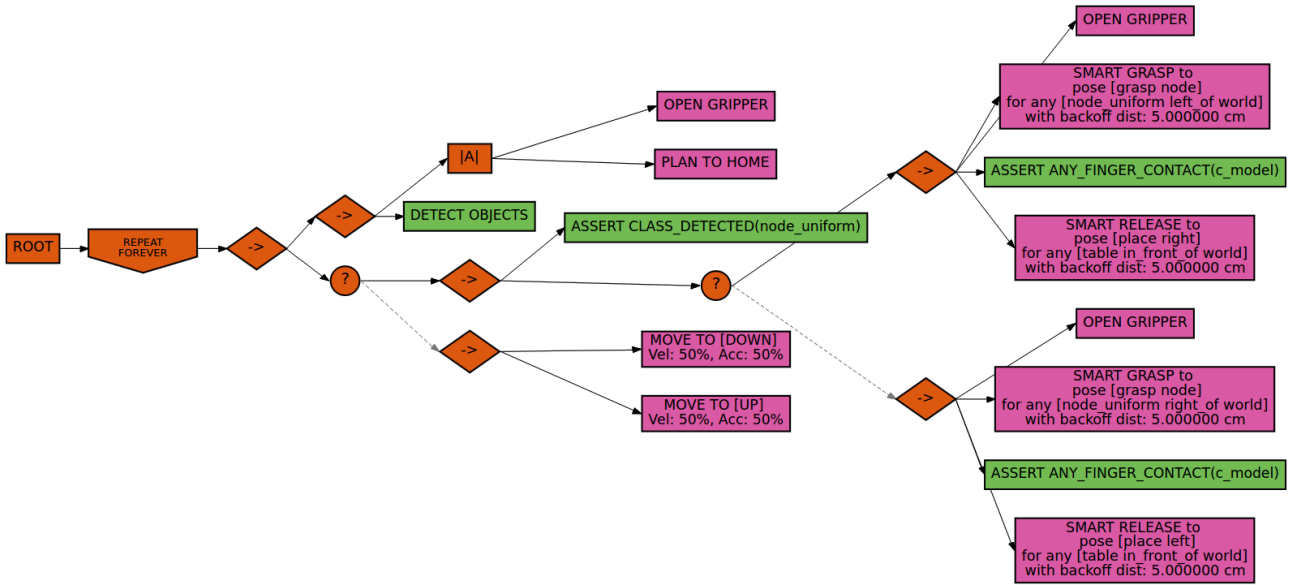


Fig. 2: A sample CoSTAR Behavior Tree displaying a complex task plan, including error checking and high-level queries to pick up and manipulate objects. Colors have been altered for readability.

These elements can be combined to create relatively complex behavior. Fig. 2 shows a tree that moves to the `Home` joint position and opens the gripper in parallel, detects all objects in the world, and then checks to see if it found any objects of class `Node`.

B. CoSTAR Architecture

CoSTAR has a modular, component-based architecture. What follows is a brief high-level overview of the CoSTAR system; for more detail see Paxton et al. [5]. In particular, we will refer to **Operations** *u*, which are specific actions that influence the world or update the robot's knowledge thereof. These are generally exposed as BT leaf nodes such as manipulation actions or calls to object detection software. Operations can update the value of **Symbols** such as waypoints or known object positions, and also expose information as **Predicates** that make logical statements about the world like "object A is to the left of object B."

The main components of the CoSTAR system are shown in Fig. 3. In this work, we add the `SmartGrasp` and `SmartRelease` operations, building on previous functionality, to allow users to better perform pick and place tasks.

1) Target Users: Modern manufacturing is increasingly a high-tech field that requires strong problem-solving, math, and technical skills in its workers [1]. As such, CoSTAR is designed for expert workers with a STEM background, though not necessarily with formal schooling.

2) User Interface: The heart of the CoSTAR UI is the BT-based task editor, which allows the end user to combine and parameterize operations exposed to the user by all of these different components. Users can switch the robot into a compliant mode by pressing a `TEACH` button, and can enable autonomous execution by pressing the `SERVO` button. Other functionality is hidden in a collapsible set of menus in the lower right, which bring up dialogs for saving waypoints or SmartMove positions. The task editor is accompanied by a 3D visualization of the robot, detected objects, and coordinate frames via the ROS RViz interface [20].

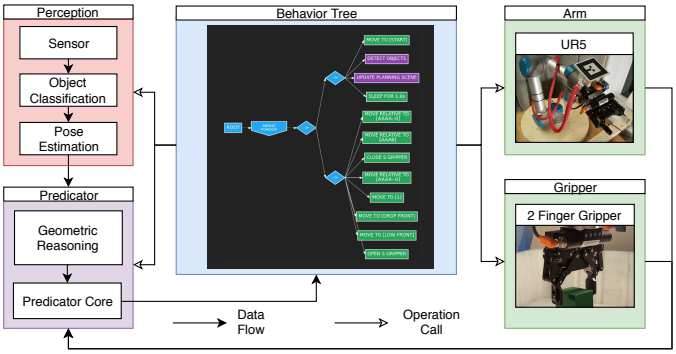


Fig. 3: A compressed overview of the CoSTAR system, based on the similar diagram in our previous work [5].

3) Perception and System Knowledge: CoSTAR stores knowledge accumulated from a distributed set of sensors and components through a special component called `Predicator`. Predicator aggregates information about which objects have been detected, what types of objects they are, and how they relate to each other and the task at hand. It is exposed to end users through the `KnowledgeTest` and `PoseQuery` operations. The `KnowledgeTest` operation checks to see if a certain predicate is true. In Fig. 2, we use knowledge tests to make sure that a `node` object has been found, and to make sure that the gripper successfully grasped an object. One specific source of knowledge is the `DetectObjects` operation, which calls a 3D perception system to detect and estimate 6DOF poses for all objects in the scene [5]. For example, the `DetectObjects` operation

appears near the top of the tree in Fig. 2. In this work, we also add a `DisableCollisions` operation, which allows us to specify that the robot can move closer than its minimum safe distance to a particular object (e.g. to pick it up or push it).

4) Motion Planning and Execution: The `Arm` component handles motion planning and execution, and ties in closely with the `Predicator` component to expose more advanced operations. Grasping an object requires multiple steps: the robot must (a) align the gripper properly with the grasp position, (b) move in, (c) close the gripper, and finally (d) lift the object. The steps to this simple task plan remain the same in a wide variety of tasks, being parameterized solely by the grasp frame and the backoff distance. This backoff distance is used in steps (a) and (d) to compute how far back from the object the robot will position its gripper. Placing an object is similarly complex.

In this work, we propose two unique actions to make pick and place tasks easier to specify in a general way: `SmartGrasp` and `SmartRelease`. Each solves a planning problem for either grasping or placing with multiple possible goals based on user-provided information. As a result, these two `SmartMoves` are more intuitive and reliable than the capabilities found in previous versions of CoSTAR [5]; where operations had no memory of the specific object being manipulated and thus sequential motions occasionally led to unexpected behavior.

`SmartGrasp` and `SmartRelease` query the `Predicator` component for objects matching some set of conditions, and use object symmetry information to generate a list of potential grasps. Then the `Arm` computes backoff poses and sorts them based on joint-space distance. The resulting sorted list of grasp poses is used to generate motion plans in order of preference via the RRT-Connect algorithm [21] so that it will grab the closest object that meets the given criteria. In effect, users can then frame the task plan as a sequence of high-level commands such as:

```
grasp(obj) with grasp_position such that
is_node(obj) and right_of(robot, obj)
```

to grasp any `node` object on the right side of the robot.

`SmartRelease` is largely the same as `SmartGrasp`, but it computes the backoff pose in the world frame's $-z$ axis and opens the gripper. This is helpful when stacking objects or placing them on a table. The backoff distance can be tuned by the user to achieve different behaviors. Again, the `SmartRelease` is parameterized by a single demonstrated action and a predicate specification.

IV. User Study

The goal of this study was to see how well users could use the operations provided by each of four different perception strategies in order to adapt to an increasingly complex task.

Participants were randomly assigned to one of four groups, each associated with a specific level of system capabilities and then asked to complete the study tasks. Fig. 4 compares resulting task plans. This procedure was approved by the Johns Hopkins University Institutional Review Board (IRB) under protocol #HIRB00005268.

Condition 1 — *Simple* represents a system similar to the commercially available programming environments of the Universal Robot UR5. The robot can open or close its gripper and move to positions relative to the robot base, but does not have access to any other CoSTAR capabilities including motion planning.

Condition 2 — *Motion Planning* adds motion planning: the user can update the scene model by detecting objects, but perception is only used to update scene geometry for the purposes of object avoidance. Thus, the robot will plan trajectories that avoid collisions and joint limits, but there is no ability to move relative to detected objects. The `DisableCollisions` operation allows the robot to move close to a specific object by disabling collision checking against it. This condition represents a "naive" approach of motion planning without abstraction.

Condition 3 — *Relative Motion* provides simple perception-based movements: users have access to the `DetectObjects` operation and can define waypoints relative to detected object positions. This represents the "naive" approach to incorporating object pose information where users must explicitly handle every object in the scene.

Condition 4 — *SmartMove* exposes two types of high-level actions: `SmartGrasp` and `SmartRelease`. They were asked to use these capabilities and the `PlanToHome` action to complete the task. These represent abstract, high-level set of capabilities instead of the simpler, more explicit capabilities exposed in the other conditions.

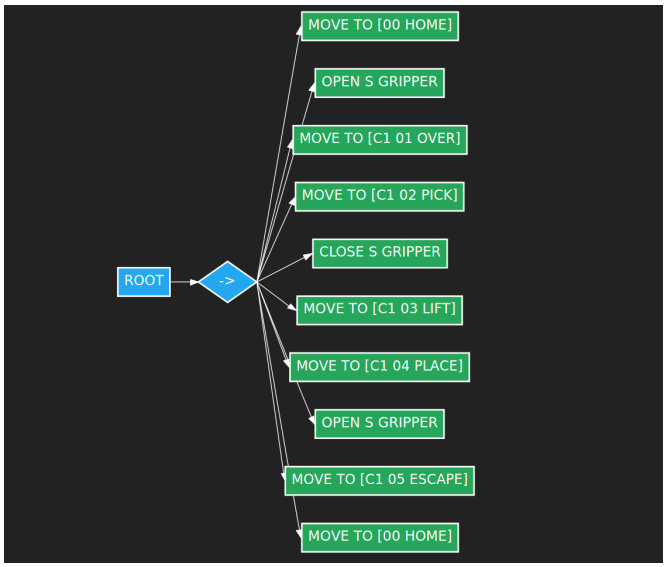
A. Study Tasks and Procedure

We broke the study up into three phases: training, task performance, and the exit survey with subsequent generalization experiments on the user created trees.

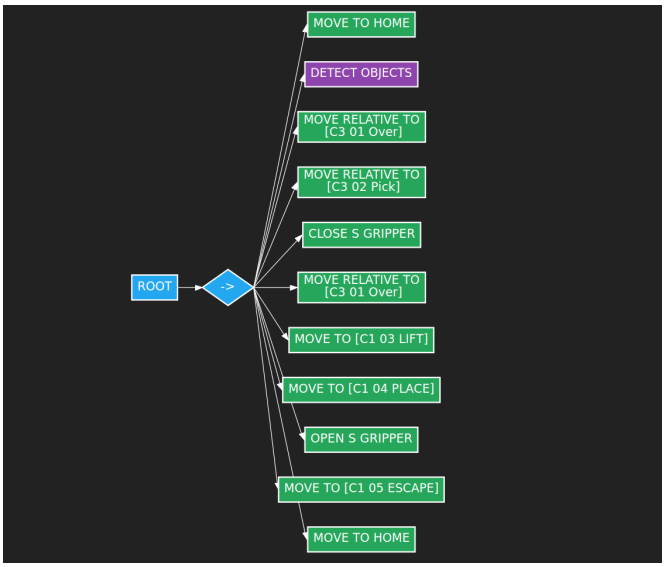
Phase 1: We began with a short training phase, in which the participant moved a single block from the right to the left side of the robot. This task was not timed or used to score participants, and users could ask as many questions as they wanted. Users were given an opportunity to ask any questions before moving on. This phase took 15 minutes.

Phase 2: Next we presented participants with three pick-and-place tasks with increasing complexity. They were given 15 minutes to complete as much of each task as possible. These tasks were designed to enable participants to incrementally learn and put into practice how each UI component works, how the robot responds to user commands, and how to build task plans using specific technologies such as perception and planning. All three tasks required that participants move square blocks called "nodes" in different configurations from the right to the left of the robot without knocking over an obstacle: a red "link."

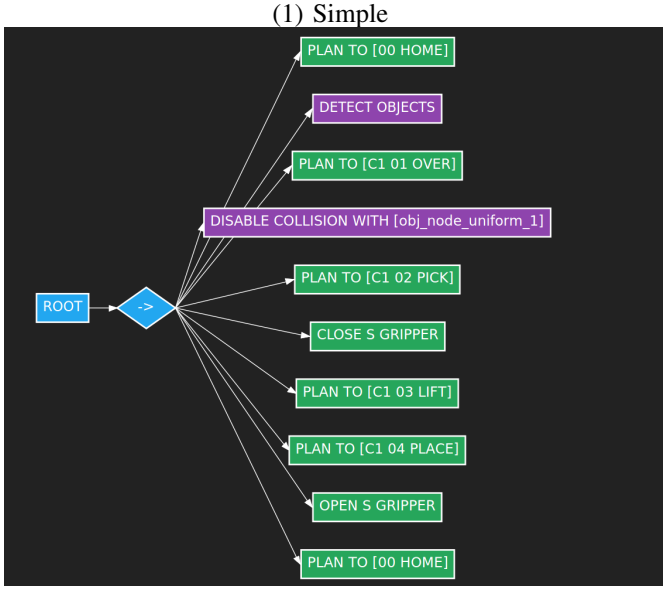
Task A — *Move Blocks* asked participants to move two blocks from the right side of the workspace to the left. Participants had to apply the knowledge from Phase 1 to teach the robot to move the second block themselves. The obstacle was introduced to the world, but far enough away



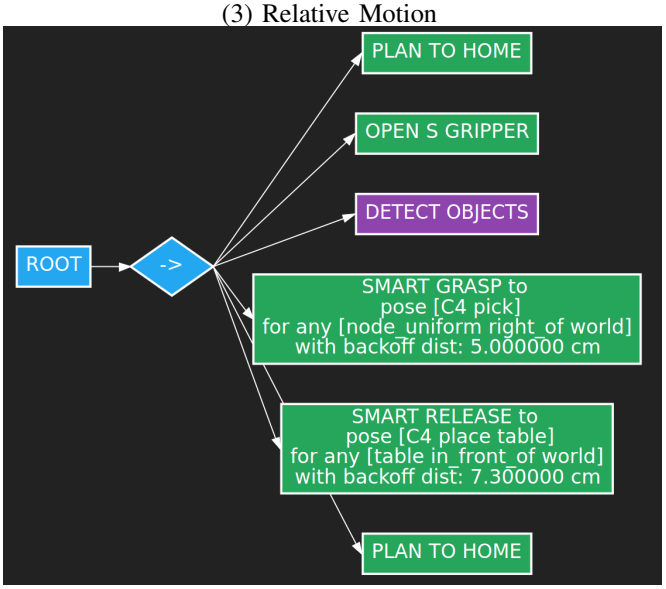
(1) Simple



(3) Relative Motion



(2) Motion Planning



(4) SmartMove

Fig. 4: Different CoSTAR operations were enabled and disabled under four different conditions to test different strategies for incorporating perception, as described in Section IV. All versions used the same BT-based user interface. Waypoints are expressed in square brackets (e.g. `C1 01 OVER`). Likewise, object names and parameters of `SmartMoves` appear in their respective boxes.

from the blocks that participants did not need to actively avoid it.

Task B — *Avoid Obstacle* required participants to similarly move two blocks from the right to the left, although one of the blocks was placed in a different position from the previous task and the obstacle was placed closer to the two objects (Fig. 5a).

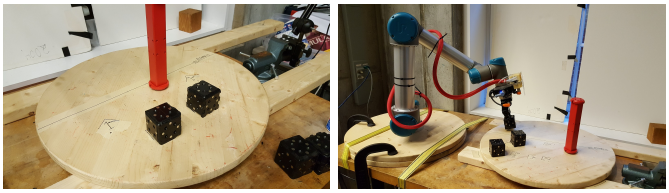
Task C — *Place Link* presented participants with three blocks all of which were in positions different from previous tasks. The link was moved farther away again, and participants were asked to move two blocks of their choice and to pick up the link and place it on top of one of these blocks. This configuration is shown in Fig. 5b.

Phase 3: Finally, participants filled out a questionnaire that included the System Usability Scale [22] and answered a set of interview questions. After they left, we tested the generalization of user-authored task plans in two different cases , shown in Fig. 6. In one case, we add additional obstacles to a scene and compare the motion planning vs. SmartMove condition. In the second case, we move objects to new positions to determine if SmartMoves can adapt better than Relative Motions to the new positions.

B. Hypotheses and Metrics

We examine three specific hypotheses in this study:

H1. CoSTAR's SmartMove system will be perceived to be at least as usable as the baseline,



(a) (b)

Fig. 5: Two of the study tasks. In Fig. 5a, users move two blocks from right to left; in Fig. 5b users can use any two blocks and additionally were asked to pick up the red link.

Fig. 6: Alternate world configurations. In the first case, additional links are added to Task 3 as obstacles. In the second, all objects in Task 3 are moved to new positions.

H2. When users are asked to generalize their Task 1 plan to a new environment configuration, users with the SmartMove condition will perform best.

H3. SmartMove plans will likewise perform best on generalization tasks

We developed a scoring metric to compare user performance. Users can attain a maximum of 1 point per component of the final assembly, with a bonus for fast task completion. Performance on Task 1 and Task 2 was scored according to: $score = n_{nodes} - n_{errors} + 2 \times (1-t)$, where t is the fraction of time in the trial used between 0 and 1, n_{nodes} is the number of nodes successfully moved from the right to the left, and n_{errors} is 1 if the link was knocked over and 0 otherwise. This score was configured to give a bonus to users who completed the task quickly, with a strong score of 3 if they used half of their time to finish the task.

We scored performance on Task 3 based on how many blocks were moved. The Behavior Trees created by users were retained and tested after the end of the trial to compute this score. The user received (1) one point for each of up to two nodes moved and (2) one point if the link was successfully moved and placed. Due to the difficulty of using a novel component, we give partial credit of 0.25 points for attempting to grasp the link, 0.75 points for moving and placing the link but not achieving a successful mate, with 1.0 point reserved for a perfect task performance. We give the same time-to-completion bonus as for Task 2 above.

We quantify perceived usability with a variant of the commonly used System Usability Scale (SUS) [22], which has been found to correlate well with other metrics for usability [23]. Table I shows the items included in the scale.

TABLE I: Modified System Usability Scale for robotics user interface. Responses are scored from 1 (strongly disagree) to 5 (strongly agree).

#	Statement
1	I think that I would use this interface frequently.
2	I found this interface unnecessarily complex.
3	I thought the interface was easy to use.
4	I would need the help of a robotics expert to be able to use this interface.
5	I found the various functions provided by this interface to be well integrated.
6	I thought the interface design was too inconsistent.
7	I imagine most people would learn to use this interface very quickly.
8	I found this interface cumbersome to use.
9	I feel very confident in using this interface.
10	I need to learn a lot of things before I could be an effective user of this interface.

C. Participants

Our target demographic represents highly skilled manufacturing workers with a STEM background [1]. To approximate this demographic, we performed our study on undergraduate and graduate students in math, science, engineering, and computer science: users who are technically savvy with some education but who are not necessarily experts in robotics. Subjects were recruited via email.

We recruited at least 8 users for each of 4 conditions, in line with accepted practices for studies in this category[24]. Altogether, 40 users participated in the study. Five users were excluded for the following reasons: three were excluded for being non-technical novice users and not undergraduate or graduate students, and two users were excluded due to issues with the perception system during their trials. Characteristics of the population are described in Table II. Our user pool is less familiar with robotics ($M = 3.5$, $SD = 1.8$) than with programming ($M = 5.6$, $SD = 1.6$). Not coincidentally, many of our test subjects were computer science students.

D. Study Results

Data from the System Usability Scale [22] indicate that users found CoSTAR to be highly usable with an average SUS score of 73.4 out of 100 across all users. Previous work has found that a system with "good" usability will have a mean score of 71.4, while a system with "excellent" usability will have a mean score of 85.5 [23]. In aggregate, they considered CoSTAR to be good, but not a perfectly usable system.

Table III summarizes the task performance and usability data collected in the study. We limited the time users were able to spend on each task in order to see some variability in task success rates, since all versions of the system were capable of completing all tasks given enough time. In general, users performed similarly on Task A, as seen in Fig. 7.

Task B: SmartMove users performed significantly better than Motion Planning users on Task B ($p = 0.0096$) or Relative Motion users ($p = 0.0079$) according to a pairwise T-test on our task completion metric. They only performed marginally better than users of the Simple condition ($p =$

TABLE II: Self-reported participant population characteristics.

Population Characteristic		Value
Average Age		23.0 ± 3.1
Gender	# Male	25
	# Female	10
Familiarity (1 to 7)	Robotics	3.5 ± 1.8
	Programming	5.6 ± 1.6
	Video Games	3.8 ± 1.8
Major	Computer Science	15
	Robotics	4
	Other Engineering	7
	Other Math/Science	2
	Unspecified	7

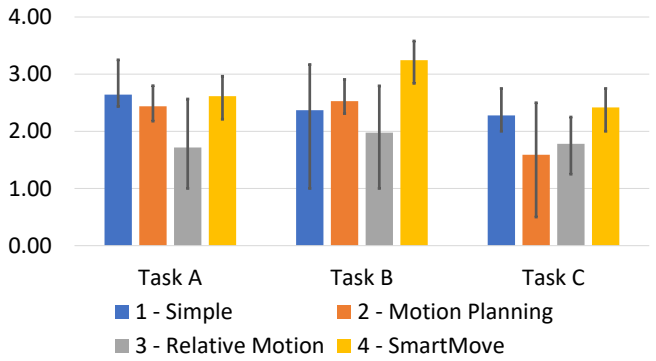


Fig. 7: Scores for each condition on tasks A-C. Error bars indicate interquartile range. Results showed a wide variance of final scores for Task C in particular; users had trouble understanding how best to use many variants of the system.

0.0207). All significance results use Bonferroni-corrected alpha levels of $0.05/3 = 0.0167$.

Task C: User performance varied the most on Task C, most likely due to the difficulty of learning how to manipulate a new object. Scores from SmartMove are still higher than those from the Motion Planning and Relative Motion groups ($p = 0.035$ and $p = 0.048$, respectively). Differences are even clearer when comparing total scores across all three tasks as shown in Table III: SmartMove is better than both Motion Planning ($p = 0.007$) and Relative Motions ($p = 0.003$).

Generalization: The SmartMove system also had a higher score than Motion Planning at the generalization task with extra obstacles ($p = 0.029$). SmartMove scored an average of 1.65 ± 0.89, Motion Planning scored an average of 0.43 ± 1.28. On the generalization task with changed object positions, it likewise scored higher than the Relative Motion test case ($p = 0.056$), with an average of 2.25 ± 0.35 vs. 1.5 ± 1.29. We saw much more variation on the object pose generalization task for the Relative Motions task: some users' trees generalized very well, others very poorly. SmartMove task plans were comparatively very consistent across users.

V. Discussion

There are three key findings from this study. First, these results show that users find Behavior Trees to be a practical and effective means of defining a robot program. Second, perceptual abstractions such as SmartMove allow end users to more easily specify and adapt robot programs. Finally, such programs are more general and more robust to environmental variation. Users can employ advanced robotic capabilities if exposed to them properly and with proper training. For motion planning and perception to make a substantial impact they must support the user's mental model of the task.

Users were comfortable with the mixture of hands-on teaching and editing the Behavior Tree, as shown in Fig. 1. Participants assigned to the Simple or Motion Planning groups found the robot easy to manage and predictable, but they expressed frustration by the degree to which they had to micro-manage positioning the robot by specifying multiple waypoints. One user operating under the Simple condition complained that in a large tree, "replacing them one by one every time is a little inconvenient." The effect was clear when users were asked to generalize their plan for Task 1 to solve Task 2: SmartMove users clearly outperformed the others, lending strong support to H1. Indeed, some users of the Simple condition struggled with this generalization because they had so many waypoints that needed to change.

We found evidence for H2, noting that our perceptually abstracted SmartMoves offered superior generalization to Motion Planning or Relative Motion alone. When tested on more challenging scenarios with additional obstacles, trees using SmartMove were able to adapt by choosing an alternate grasp or an alternate object to pick up when a more explicit method for specifying the task would have failed. The Simple condition, while easy to use, had no ability to generalize whatsoever.

Perception was not useful to our participants if they could not communicate effectively with the robot. Compare the Simple condition with Motion Planning and Relative Motion in Fig. 7. It offers no built-in problem solving ability (resulting in high variance in Task B) but is comparable or better than these conditions. Users of Motion Planning and Relative Motion often could not predict what the robot would do or why it would do it. As a result, these were less effective or consistent than either the SmartMove or Simple conditions. Participants were often unclear on when knowledge they provided to the robot would generalize and when it would not. One user assigned to the Relative Motion condition felt the robot "spazzed out," i.e., it did not do what they were expecting.

SmartMove was more predictable, although there is room to improve the way the design and implementation details are communicated within the interface. For instance, users often believed that `SmartGrasp` operations had to be re-taught for every node object, but this is not necessary. On the other hand, it was also not clear that new drop positions must be taught to `SmartRelease` for every new node in the workspace. The physical meaning of the arguments for the SmartMove query could also benefit from improved clarity. For example, One user said they "were not able to figure out why [the next block] was left," they only knew that it worked. These points suggest areas for future improvement.

TABLE III: Results from each of the four conditions on the three tasks showing performance times and self-reported System Usability Scale (SUS) scores after completion of the experiment.

		Task A			Task B			Task C		Total
Condition	# Users	Score	% Success	Time	Score	% Success	Time	Score	SUS	Score
1 Simple	9	2.64 ± 0.73	88.9%	8:39	2.37 ± 1.04	66.7%	7:06	2.28 ± 0.52	73.8 ± 8.7	7.15 ± 1.78
2 Motion Planning	9	2.44 ± 0.32	100.0%	11:43	2.53 ± 0.66	88.9%	9:36	1.59 ± 1.13	74.2 ± 10.1	6.58 ± 1.67
3 Relative Motion	8	1.72 ± 0.97	71.4%	12:24	1.98 ± 1.13	62.5%	9:16	2.1 ± 1.24	69.5 ± 10.4	5.56 ± 1.96
4 SmartMove	9	2.62 ± 0.35	100.0%	10:23	3.24 ± 0.47	100.0%	5:40	2.42 ± 0.50	75.3 ± 6.7	8.31 ± 0.34

VI. Conclusions

In conclusion, our results show that abstract perception allows users to construct robust, generalizable task plans without negatively impacting usability. In addition, behavior Trees can form the basis for a powerful, flexible, and highly usable visual programming language for collaborative robots.

ACKNOWLEDGMENT

This work was funded by NSF Award No. 1637949.

References

[1] C. Giffi, B. Dollar, M. Drew, J. McNelly, G. Carrick, and B. Gangula, "The skills gap in US manufacturing 2015 and beyond," *Washington, DC: Deloitte Development LLC*, 2015.

[2] H. Nguyen, M. Ciocarlie, K. Hsiao, and C. C. Kemp, "ROS commander (ROSCo): Behavior creation for home robots," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 467–474.

[3] C. Mateo, A. Brunete, E. Gambao, and M. Hernando, "Hammer: An Android based application for end-user industrial robot programming," in *Mechatronic and Embedded Systems and Applications (MESA), 2014 IEEE/ASME 10th International Conference on*. IEEE, 2014, pp. 1–6.

[4] S. G. Brunner, F. Steinmetz, R. Belder, and A. Dömel, "RAFCON: a graphical tool for task programming and mission control," *arXiv preprint arXiv:1605.09185*, 2016.

[5] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, and G. D. Hager, "CoSTAR: Instructing collaborative robots with behavior trees and vision," *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, 2017.

[6] D. K. Misra, J. Sung, K. Lee, and A. Saxena, "Tell me dave: Context-sensitive grounding of natural language to manipulation instructions," *Proceedings of Robotics: Science and Systems (RSS), Berkeley, USA*, 2014.

[7] S. R. Ahmadzadeh, A. Paikan, F. Mastrogiovanni, L. Natale, P. Kormushev, and D. G. Caldwell, "Learning symbolic representations of actions from human demonstrations," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 3801–3808.

[8] S. Levine, N. Wagener, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 156–163.

[9] I. Dianov, K. Ramirez-Amaro, P. Lanillos, E. Dean-Leon, F. Bergner, and G. Cheng, "Extracting general task structures to accelerate the learning of new tasks," in *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*. IEEE, 2016, pp. 802–807.

[10] C. Paxton, F. Jonathan, M. Kobilarov, and G. D. Hager, "Do what I want, not what I did: Imitation of skills by planning sequences of actions," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 3778–3785.

[11] H. Sharp, Y. Rogers, and J. Preece, *Interaction Design: Beyond Human Computer Interaction*. John Wiley & Sons, 2007.

[12] M. Beetz, F. Bálint-Benczédi, N. Blodow, D. Nyga, T. Wiedemeyer, and Z.-C. Márton, "Robosherlock: unstructured information processing for robot perception," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1549–1556.

[13] J. A. Bagnell, F. Cavalcanti, L. Cui, T. Galluzzo, M. Hebert, M. Kazemi, M. Klingensmith, J. Libby, T. Y. Liu, N. Pollard, *et al.*, "An integrated system for autonomous robotics manipulation," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 2955–2962.

[14] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ogren, "Towards a unified behavior trees framework for robot control," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 2014.

[15] D. Hu, Y. Gong, B. Hannaford, and E. J. Seibel, "Semi-autonomous simulated brain tumor ablation with raven-ii surgical robot using behavior tree," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3868–3875.

[16] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *International Joint Conference on Artificial Intelligence*, 2015.

[17] N. Dantam, P. Koine, and M. Stilman, "The motion grammar for physical human-robot games," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 5463–5469.

[18] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson, "Efficiently combining task and motion planning using geometric constraints," *The International Journal of Robotics Research*, vol. 33, no. 14, pp. 1726–1747, 2014.

[19] K. R. Guerin, C. Lea, C. Paxton, and G. D. Hager, "A framework for end-user instruction of a robot assistant for manufacturing," in *Proceeding of IEEE International Conference on Robotics and Automation*. IEEE, 2015.

[20] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[21] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 995–1001.

[22] W. Albert, T. Tullis, and D. Tedesco, *Beyond the usability lab: Conducting large-scale online user experience studies*. Morgan Kaufmann, 2009.

[23] A. Bangor, P. Kortum, and J. Miller, "Determining what individual SUS scores mean: Adding an adjective rating scale," *Journal of usability studies*, vol. 4, no. 3, pp. 114–123, 2009.

[24] K. Caine, "Local standards for sample size at CHI," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 2016, pp. 981–992.