Supporting End-User Creation of Robot Task Plans for Flexible Manufacturing

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? In this work, 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 baseline versions of CoSTAR. Overall, participants found CoSTAR in general 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. They showed better performance vs. baseline methods 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, todays 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.

There is substantial interest in making it easy for domain experts to transfer knowledge to collaborative robots, either through a user interface [3], [4], [5], [2], 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]. This is a problem, as people tend to think of robot motions abstractly, e.g., "grab the next available component for this assembly," instead of as explicit 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 should allow expert users to author complex plans for collaborative robots. We previously demonstrated that CoSTAR allows users to solve a wide variety of problems by applications to tasks including sanding, assembly, and wire bending [2].

In this paper, we explore the trade-offs in managing increasingly complex robot capabilities and system behavior through a user study in which expert users instruct a collaborative robot on structure assembly tasks involving the manipulation of a set of magnetically-latching blocks. In particular, we examine the usability of CoSTAR's use of Behavior Trees and whether or not perceptually-abstracted SmartMoves makes for a better user experience and higher task performance.

[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 Science, 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

To represent different strategies for incorporating advanced capabilities such as perception and motion planning, we compare four versions of the CoSTAR system: (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, and (3) *Relative Motion*: a system that can detect object positions but requires that users explicitly specify how to interact with each object. We compare these to condition (4) *SmartMove*, which integrates perception and planning via abstract queries for objects matching a specified predicate. Users found CoSTAR and Behavior Trees to be a usable and even enjoyable system; we also found that the higher levels of abstraction present in condition (4) allowed for improved task performance and better generalization to new environments[4].

II. Background

There is strong interest in developing systems and user interfaces that allow non-expert users to program robots. Recent approaches relevant to the robotics domain include both new user interfaces [3], [4], [5], learning from demonstration [7], [10], or systems that make use of natural language together with onotologies and large knowledge bases to follow high-level instructions, like 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 [3], [5], [2]. Nguyen et al. [3] describe ROS Commander as a user interface based on finite state machines for authoring task plans. Similarly, Steinmetz and Weitschat [5] 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 allows planners to integrate selecting parameterizations of a task 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].

[4]Supplementary video of experiments and of the CoSTAR system in use is available on YouTube, showing expert instruction of the CoSTAR system and highlights from trials: `https://www.youtube.com/playlist?list=PLF86ez-NVmyFMuj10dkUkgGlGpcM5Vok9`

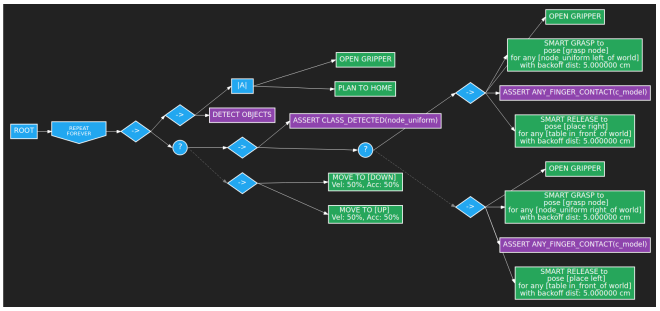


Fig. 2: A sample CoSTAR behavior tree displaying a complex task plan, including error checking and high-level queries to pick up and manipulate objects.

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 [2].

The underlying task is represented as a Behavior Tree (BT). BTs allow us to visually construct complex behaviors concurrently that are 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.

A. Overview of Behavior Trees

CoSTAR's behavior tree contains standard behavior tree elements, which are classified as follows:

- Root: Marks the start of the behavior tree.
- Operations: leaf nodes that affect the state of the world. These can test predicates, update planning information, or send commands to hardware, and communicate directly with CoSTAR's various components.
- Logical nodes: internal nodes that determines the order in which operations are performed.
- Decorators: internal modes that have only one child, which performs a predetermined action depending on the return value of the child. Repeat, Reset, and Wait for Success action are available in our system.

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 – it keeps all parts of the tree synchronized.

Complex task structure is achieved via **Logical** nodes. These control program flow and order of operations. Our logic nodes include:

- Sequence (`->`): Tick all children in order. Will stop when a tick reaches a running child.
- Selector (`?`): Try 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.

Decorator nodes include the Repeat node, which will tick its child some (possibly infinite) number of times. Many of these nodes are shown in Fig. 2.

These elements can be combined to create relatively complex behavior. In Fig. 2, we show a tree that (1) moves to the `Home` joint position and opens the gripper in parallel, (2) detects all objects in the world, and then (3) 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. [2]. In particular, we will refer to **Operations** u, specific actions that influence the world or update the robot's knowledge thereof. They 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."

CoSTAR stores knowledge from a distributed set of sensors and components through a special component called `Predicator` which stores and tracks the robot's knowledge of the world. Predicator stores information on 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 [2].

The `Arm` component handles motion planning and execution, and ties in closely with the `Predicator` component to expose more advanced operations. In addition to a variety of basic movements relative either to an object or the robot's coordinate frame, we add the `SmartGrasp` and `SmartRelease` actions. These high-level actions query the `Predicator` component for objects matching some set of conditions. The `Arm` then uses the resulting sorted list of grasp poses to generate motion plans in order of preference via the RRT-Connect algorithm. 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.

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: (1) simple, (2) motion planning, (3) relative motion, and (4) SmartMove, and then asked to complete the study tasks. Examples of resulting task plans for each condition are shown in Fig. 3.

Condition 1 - *Simple* represents a system similar to the commercially available programming environments of the Universal Robot UR5. The robot can use its gripper or servo to positions relative to the robot base, but do not have access to any other CoSTAR capabilities.

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 an object. Also, participants must use the `DisableCollisions` operation with the ID of any object they wish to manipulate to remove it from the motion planning scene.

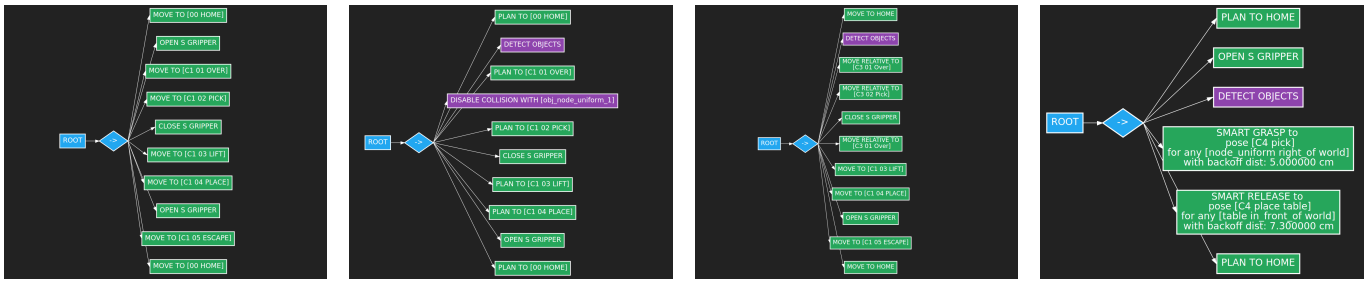
Condition 3 - *Relative Motion* provides simple perception-based movements: users have access to the `DetectObjects` operation, and can define waypoints relative to detected object positions. This represents the "naive" approach to incorporating perception: without high-level information, users must explicitly handle every object in the scene.

Condition 4 - SmartMove exposes two types of high-level actions: the `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

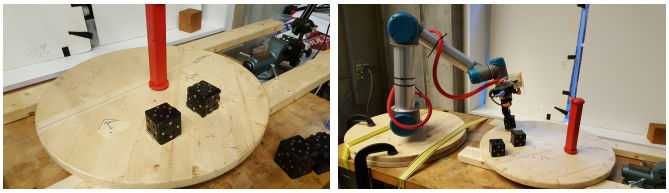
We broke the study up into three phases: training, task performance, and finally the exit survey and generalization experiments.

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. After this training period, users were asked to perform Tasks A, B, and C. Users were given an opportunity to ask any questions before moving on; questions were not timed.



(1) Baseline (2) Motion Planning (3) Relative Motion (4) SmartMove

Fig. 3: The CoSTAR operations were enabled and disabled under four different conditions to test the value of each capability across users as described in IV.



(a) (b)

Fig. 4: Two of the study tasks. In Fig. 4a, users move two blocks from right to left; in Fig. 4b users can use any two blocks and additionally were asked to pick up the red link.

Phase 2: Next we presented participants with three pick-and-place tasks with increasing complexity. These were designed to enable participants to incrementally learn and put into practice how each UI component works, how the robot responded 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. The goal of this task was to determine whether participants could apply the knowledge from Task 0 to teach the robot to move the second block themselves. The obstacle was introduced to the world, but far enough away 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. 4a). As in Task A, Participants were asked to build upon their solution to the previous stage.

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. 4b.

Phase 3: Finally, participants filled out a questionnaire that included the System Usability Scale [19] and answered a set of interview questions.

Afer they left, we tested the generalization of user-authored task plans in two different cases. 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.

This procedure was approved by the Johns Hopkins University Institutional Review Board (IRB) under protocol #HIRB00005268.

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, (H2) when users are asked to generalize their Task 1 plan to a new environment configuration, users with the SmartMove condition will perform best, and (H3) their 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 percentage 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. Created Behavior Trees were retained and tested after the end of the trial to compute this score. The user received one point for (1) each node moved (up to two total points), (2) one point if the link was successfully moved and placed. Due to the difficulty of using a novel component, we give partial credit for 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) [19], which asks users a set of 10 questions to rate their agreement with statements such as "I think that I would use this interface frequently." SUS has been found to correlate well with other metrics for usability [20]. 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.

Altogether, 40 users participated in the study. Three were excluded for being non-technical novice users, not undergraduate or graduate students; two final users were excluded due to issues with the perception system during their trials. 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.

V. Results

Data from the System Usability Scale [19] indicate that users found CoSTAR to be highly usable. 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 [20]. In aggregate, they considered thought CoSTAR was good, but not a perfectly usable system.

Table I 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. 5.

Task B: SmartMove users performed significantly better than Motion Planning users on Task B ($p = 0.0096$) or Relative Motion users ($p = 0.0079$). They only performed marginally better than users of the Simple condition ($p = 0.0207$) based on 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. We 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 I: 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 Planing an average of 0.43 ± 1.28. On the generalization task with changed object positions, it

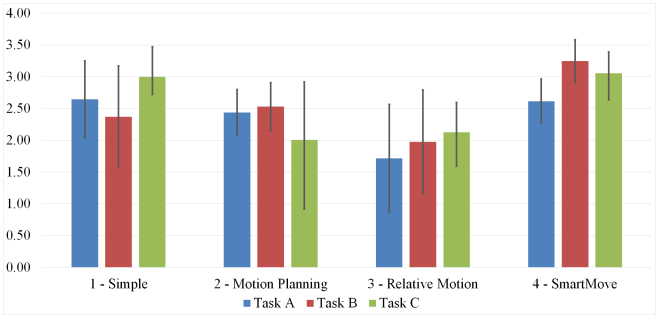


Fig. 5: Comparison of scores for each condition on tasks A-C. Results showed a wide variance of final scores for Task C in particular.

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.

VI. Discussion and Conclusion

There are key three 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.

Despite being given very little time to acclimate to the system, users quickly learned to solve problems. They were comfortable with the mixture of hands-on teaching and editing the Behavior Tree; Fig. 1 shows two examples of users interacting with the CoSTAR system. We see high perceived usability across all variants of the system, with the highest scores reserved for the SmartMove condition, which provides strong support for H1. In addition, performance metrics on Task B and Task C provide strong support to H2.

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 Simple user complained that in a large tree, "replacing them one by one every time is a little inconvenient." The effects of this difficulty were clear when users were asked to generalize their plan for Task 1 to solve Task 2: SmartMove users clearly outperformed the others.

Most of the time users spent attempting to generalize their SmartMove trees was either (1) thinking about the tree, trying to decide if they needed to change anything; or (2) adjusting a node's drop position if it was too close to the new obstacle. Since drop positions were still manually specified, this means that most challenging part of using SmartMoves was the part that was most similar to the other conditions.

TABLE I: 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 Baseline	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

We found evidence for H3, noting that our perceptually abstracted SmartMoves offered superior generalization to Motion Planning or Relative Motions 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 system, while easy to use, has no ability to generalize whatsoever.

Users often could not predict what the robot would do or why it would do it. This contributed to making the Motion Planning and Relative Motion conditions less effective than the Simple or SmartMove 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. Usually this occurred because an object had been assigned a new ID by the perception algorithm.

SmartMove was not unpredictable to the same extent, though many users still had trouble understanding details of its implementation. One particularly common misunderstanding was that `SmartGrasp` operations had to be re-taught for every object of the node class and that `SmartRelease` operations did not need to be re-taught for every new node in the workspace. Other users had trouble understanding the arguments for the SmartMove query; one user say they "were not able to figure out why [the next block they wanted to grab] was left," they only knew that it worked.

Our results show that abstract perception allows users to construct robust, generalizable task plans. It does not negatively impact usability and allows for easier adaptation to new environments and problems. In addition, Behavior Trees can form the basis for a powerful, flexible, and highly usable visual programming language for collaborative robots.

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] 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 (to appear)*, 2017.

[3] 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.

[4] 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.

[5] 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.

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

[20] 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.