2017 IEEE International Conference on Robotics and Automation (ICRA)
Singapore, May 29 - June 3, 2017

CoSTAR: Instructing Collaborative Robots with Behavior Trees and Vision

Chris Paxton, Andrew Hundt, Felix Jonathan,
Kelleher Guerin, Gregory D. Hager

Abstract— **For collaborative robots to become useful, end users who are not robotics experts must be able to instruct them to perform a variety of tasks. With this goal in mind, we developed a system for end-user creation of robust task plans with a broad range of capabilities. *CoSTAR: the Collaborative System for Task Automation and Recognition* is our winning entry in the 2016 KUKA Innovation Award competition at the Hannover Messe trade show, which this year focused on Flexible Manufacturing. CoSTAR is unique in how it creates natural abstractions that use perception to represent the world in a way users can both understand and utilize to author capable and robust task plans. Our Behavior Tree-based task editor integrates high-level information from known object segmentation and pose estimation with spatial reasoning and robot actions to create robust task plans. We describe the cross-platform design and implementation of this system on multiple industrial robots and evaluate its suitability for a wide variety of use cases.**

I. Introduction

While robots have not yet made inroads into homes or the world at large, collaborative robots work alongside humans in factories with increasing frequency. These industrial robots are common in medium and large manufacturers, but are often underutilized by small manufacturers due to the high cost of retooling and reprogramming these robots to perform a wide variety of tasks. There are two main problems with existing systems for programming these robots: clumsy user interfaces and their inability to perceive the world in ways that are meaningful to humans. Other barriers to deployment include setup time, managing configuration details, and lack of robustness to changes in the environment.

These needs have been recognized by private enterprise. KUKA Roboter GmbH posed the 2016 KUKA Innovation Award competition as the Flexible Manufacturing Challenge; indicating that *"vision, manipulation and grasping, safe and intuitive human-robot collaboration, machine learning and cloud-based operations are considered most important"* [1] to the future of the industry. Our entry, *CoSTAR: the Collaborative System for Task Automation and Recognition*, placed first among 6 finalists selected from 25 total applicants by a jury of robotics experts from industry and academia. In this work, we describe CoSTAR and how it is designed to address the demand for effective collaborative robots.

We have identified three characteristics key to a system for authoring robot task plans: *capability*, *usability*, and

This work was funded by NSF Grant 1227277. Authors are from the Dept. of Computer Science, Johns Hopkins University, Baltimore, MD, USA. Email: {cpaxton, ahundt, fjonath1, kguerin2}@jhu.edu and hager@cs.jhu.edu).

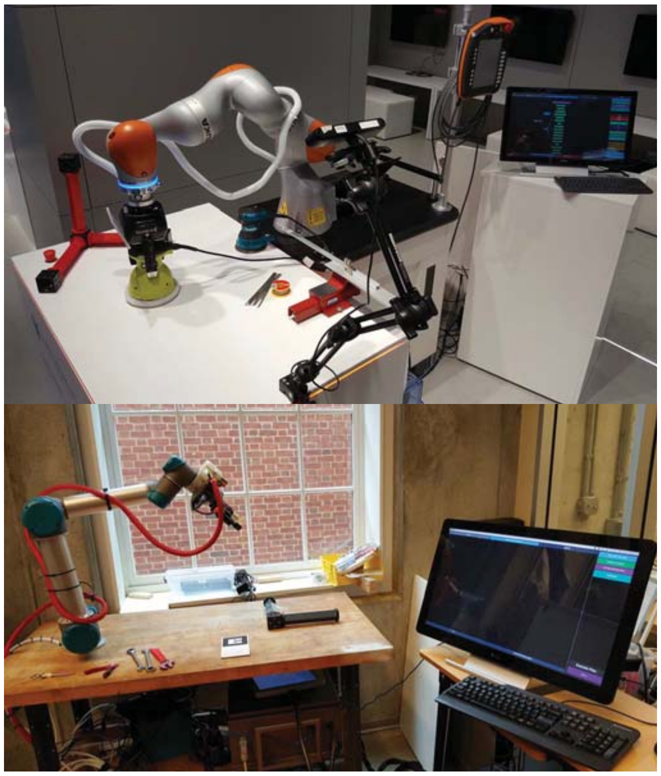

Fig. 1: The CoSTAR system set up to perform a wide variety of tasks. Top: CoSTAR running on a KUKA LBR iiwa at the Hannover Messe trade show. Bottom: CoSTAR running on a UR5 in a workshop.

robustness. First, a system should be capable of performing a wide variety of tasks. Second, end users should be able to understand the system's capabilities and efficiently create new task plans that meet their needs. Finally, task plans should be robust to variation, and repeated executions should produce the expected result. We designed CoSTAR to take these characteristics into consideration.

CoSTAR was originally proposed as a Behavior Tree based system to create task plans for industrial robots that utilize the tooling and human resources small manufacturers have available today [2]. We extended the original system into a modular, cross-platform architecture for authoring industrial robot task plans. We integrated perception with an abstracted world representation that allows end users to create task plans that are robust to environmental variation. The system is shown in Fig. 1 deployed on two different platforms.

Recent work has proposed to remedy the problems with programming robots through kinesthetic teaching methods [3], improved graphical user interfaces (GUIs) [2], [4], [5], or the use of symbols, ontologies, and natural language to enable high-level task specification [6]–[10]. We combine a powerful GUI with grounded sensor abstractions produced by CoSTAR's distributed components. While this approach requires that users be more actively involved in constructing task plans, it results in more robust and predictable task plans [4]. CoSTAR can also rely on users' domain knowledge to solve problems without a complex ontology: it is our philosophy that CoSTAR should empower end users to solve their own problems rather than providing "one size fits all" solutions.

CoSTAR is composed of *Components*, each of which is associated with input data, output data, symbols, predicates, and operations that it can perform. Input and output data are represented by ROS topics [11]. Symbols represent objects and positions, while predicates describe qualities of these objects and positions. CoSTAR includes a knowledge management component called "Predicator," which gives end users the ability to construct a wide range of different tasks that integrate state-of-the-art object detection and pose estimation proven to work in cluttered scenes [12].

Our contributions are: (1) a modular, cross-platform system designed to emphasize capability, usability, and robustness; (2) abstract perception that exposes symbols and predicates that can be used for authoring task plans; (3) evaluation of the system on a series of increasingly complex tasks; and (4) an open source implementation on a KUKA LBR iiwa 14 R820 and a Universal Robots UR5, each with different grippers.

II. Related Work

There is broad interest in making robots into intelligent, collaborative assistants that can be taught by end users to perform complex tasks [4], [5], [13]. One approach to end user collaboration is to build tasks purely from human demonstrations. This includes work which used a BP-AR-HMM to automatically segment a task into primitive actions [13]. End-to-end deep reinforcement learning has proven effective at training individual actions [14].

A second approach is to allow users to provide high-level task specifications using a domain specification language such as PDDL [7] or from natural language [8], [9]. Balakirsky et al. [7] used an ontology to perform a simple kitting task. Tenorth et al. describe KnowRob, an architecture which allows robots to share knowledge, including object models and action recipes, which can be used to accomplish a variety of household chores [6], [15]. These methods provide powerful high level descriptions, but they require a large amount of built-in knowledge from an ontology which is often found to be incomplete when attempting to create a program that solves a new problem. By contrast, CoSTAR is designed to offer a suite of basic capabilities which can be recombined to solve new tasks on the fly.

A third approach is to implement an intuitive visual user interface and allow users to construct their own solutions to tasks. Mateo et al. implement Hammer for programming industrial robots, a visual programming language based on Snap running on an Android tablet [5]. Nguyen et al. implemented ROS Commander as a system for building Hierarchical Finite State Machines (HFSMs) describing tasks for the PR2 [4]. We expand upon the Behavior Tree visual user interface described in [2].

Behavior Trees have been used to describe complex robotic manipulation tasks including a variety of object manipulation tasks [16], humanoid robot control [17], and brain tumor ablation with a Raven II surgical robot [18]. They are comparable in power to HFSMs [19], and are commonly used in the video game industry due to their superior scalability and modularity [20]. These characteristics make Behavior Trees ideal for representing collaborative robot task plans [2].

We apply the algorithm described in [12] to object detection and pose estimation in cluttered scenes. This approach is based on ObjRecRANSAC [21] with an additional step that segments objects from their environments.

III. System Design

We laid out three goals for CoSTAR: (1) it should have the capability of performing a wide variety of tasks, (2) it should be usable by non-experts, and (3) it should be robust to environmental change. Incorporating perception makes the system more capable (as perception-based tasks such as sorting are now possible) and makes the system far more robust (as tasks can be performed regardless of the movement of objects and goals). For usability, CoSTAR exposes only symbolic and qualitative information to the end user. This means that end users can formulate tasks in human terms. In the following section, we describe the modular architecture that allows us to add new system knowledge and actions to ensure CoSTAR has the capability to perform any given task. We also describe how this connects to our usability goals via a Behavior Tree-based graphical user interface. Our approach to end user task specification can be contrasted against approaches based on ontologies and symbolic task planning such as [6], [7]. Rather than relying on a large ontology, CoSTAR components define a relatively small set of geometric predicates such as `LeftOf`, `RightOf`, and `InFrontOf`, relying on end users to combine and use these symbols to specify tasks. In total we use six geometric predicates plus class identity predicates and occupancy.

A. Software Architecture

Components are an extension of the system capabilities described in our previous work [2], and are associated with a set of input data, output data, operations, predicates, and symbols. Examples of CoSTAR components include the `Perception` component described in Sec. IV, the `Gripper` component, and the `Arm` component. Consider a CoSTAR component C:

$$C = < I, O, p, s, u >$$

where $p = \{p_i\}_{i=1}^N$ is the set of predicates produced by component C, $s = \{s_i\}_{i=1}^N$ is the set of symbols produced

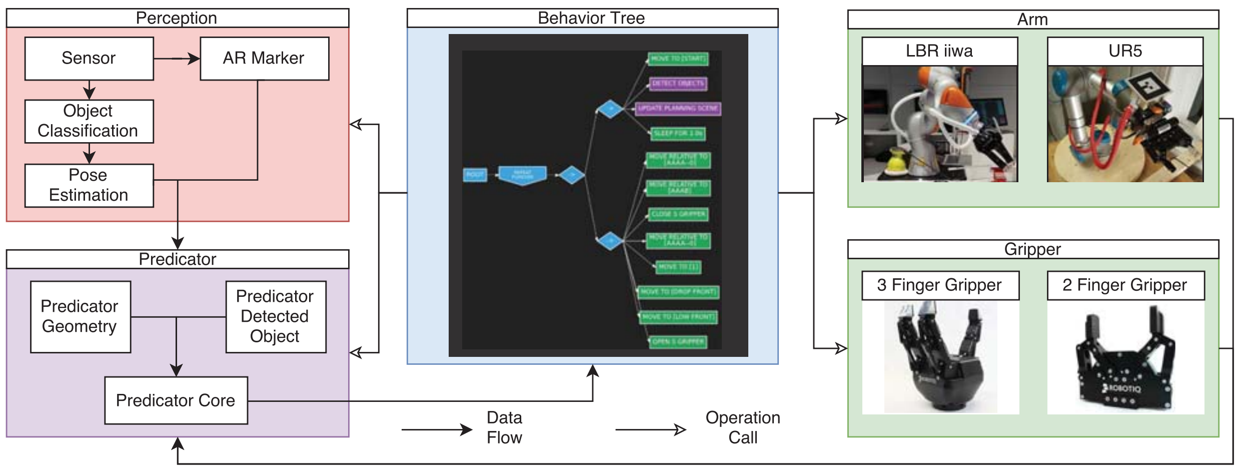

Fig. 2: Overview of the CoSTAR system. Symbols and predicates are produced by components such as Perception, Arm, and Gripper, and are aggregated by Predicator. The Behavior Tree unifies this information and uses it to encode a task, calling operations exposed by each of the individual components. Individual components can be modified or extended to add new capabilities, but the Behavior Tree only acts on abstracted information and operations directly.

by C, and $u = \{u_i\}_{i=1}^{N}$ is the set of operations made possible by C. I and O represent continuous input and output from an individual component. Different components can be concatenations of multiple sub-components, as shown in Fig. 2: continuous data flows between sub-components of `Perception` and `Predicator`. However, I and O are never explicitly exposed to the end user or the task plan. The current world state used by the Behavior Tree is wholly defined by the set of predicates and symbols produced by all of the current components. Predicates can be viewed as a special set of boolean-valued symbols that describe qualities of other symbols, and allow more complex queries such as the SmartMoves discussed in Sec. IV-B.

A key part of the modularity of the CoSTAR software design is in inheritance of components. A particular component C_p can provide a list of symbols, predicates, and operations, some of which are abstract and have not yet been implemented. All of these must be implemented by a particular instantiation of this abstract component. For an example of an abstract component see `Arm` in Fig. 2: it requires implementation of basic `Teach` and `Move` operations. They must also expose an `endpoint` symbol as a coordinate frame indicating the end of the arm. Both our `LBR_IIWA` and `UR5` components extend this `Arm` component, and implement this functionality in different ways.

Symbols s are populated from continuous input data as a function of the raw state of the world, and represent objects, positions, and object classes.

Predicates p describe qualities of existing symbols and relationships between symbols. More formally, predicates are functions

$$p(I, s_0, \ldots, s_n) \rightarrow [\texttt{TRUE}, \texttt{FALSE}]$$

that map continuous input data and a set of symbols to a boolean value. In effect, they discretize I into meaningful subsets that can be used to create generalizable task plans. Predicates are functions only of symbols and of the continuous input space: this prevents an explosion in predicate number and complexity.

Operations u are the specific actions that have an effect on the world. They can change the value of stored symbols, the state of the robot, or result in some other real-world effect. Operations typically appear as leaf nodes in the Behavior Tree; one example of an operation is `Move`.

Predicator is a special component which consists of multiple sub-components producing descriptive predicates, and provides operations allowing task plans to test the values of generated predicates. Its sub-components can be activated and deactivated according to the needs of the current system. In practice, each CoSTAR component is responsible for reporting the set of currently true predicates and the set of valid predicates and symbols to be aggregated by `Predicator`. `Predicator` then exposes operations that allow the Behavior Tree to perform queries over these predicates and symbols.

The basic components of the CoSTAR system necessary for task plan execution are `Perception`, `Gripper`, `Arm`, and `Predicator`, as shown in Fig. 2. The user interface can combine exposed operations as leaf nodes in a Behavior Tree. Additional components (such as a `PowerTool` component for turning external tools on and off in Sec. V-A) can be added as necessary.

For example, the `Gripper` component implements five operations, and also produces predicates describing the current state of the gripper. All grippers can open or close. In addition, grippers have various modes, which determine how they are going to act when being told to open or close. For the more complex adaptive gripper, these are `BasicMode`, `PinchMode`, `WideMode`, and `ScissorMode`. The parallel C-Model gripper mounted on the UR5 robot, for example, is limited to only being able to function in `PinchMode`. Any attempt to use one of the other operations would fail.

The software was implemented in ROS [11], and Orocos KDL [22] was used for computing robot kinematics as a part of the Arm component. Different components expose ROS services and topics that can be run on different machines for distributed processing and execution. The system layout, with its principal components, is given in Fig. 2.

B. *User Experience*

The goal of the CoSTAR user experience is to allow users to teach the robot naturally, through hands-on kinesthetic teaching, and to be able to create complex task plans that can be composed quickly and visually. The guiding principle is to allow users to teach a robot in the way they might teach a human, through a mixture of demonstration and explicit instruction.

As such we present users with two separate methods of interacting with the robot: (1) they can physically teach a robot to specify trajectories or learn skill models, and (2) they can interact with a graphical user interface that allows them to author a Behavior Tree. Behavior Trees are a formalism for task construction that has been previously applied to robotics in a variety of contexts [2], [16], [18] that represent tasks hierarchically. Each tree starts at a *root node* which generates ticks that propagate from left to right "down" the tree until it reaches a leaf, which will report one of `SUCCESS`, `BUSY`, or `FAILURE`. Internal nodes control the flow of operations, and send these ticks to children according to their own internal rules and state. Operations are represented by leaf nodes in the tree. In the UI shown in Fig. 3, internal nodes are blue, leaf nodes representing actions undertaken by the robot are green, and leaf nodes representing knowledge updates and queries are purple.

Internal nodes are key to creating complex task plans. Examples of internal nodes are:

- Sequence node (`->`): tick children in order, one at a time, until each one reports `SUCCESS`. If a child fails, the sequence will fail.
- Selection node (`?`): ticks children in order until one returns success. This can be used in conjunction with logical queries to determine which case in a complex program should be executed.
- Repeat node (`REPEAT N`): ticks children until N successes or failures have been reported.
- Reset node (`RESET N`): returns the same value as child, but resets the child up to N times on a failure. This has the effect of trying a child, then allowing error handling on a failure, as shown in Fig. 6.

The key advantage of behavior trees from our point of view is that they allow end users to visually create programs with the same amount of complexity and power as traditionally-written programs. Our implementation does not include (a) user specified variables or (b) variable scope. Instead, all operations must be handled within specific nodes. For a more in-depth examination of our implementation of the behavior tree formalism, see [2].

One goal of our system is to allow end users to quickly set up the platform in a new environment. To facilitate this, we added a `HandEyeCalibration` component and a corresponding *Calibrate* button to the user interface shown in Fig. 3(8). To facilitate system setup, we fix an AR marker to the end effector of the robot (visible in Fig. 2). We utilized dual quaternion hand eye calibration as implemented by CamOdoCal [23] to compute the transform from the tip of the last joint on the robot arm's model to the marker fixed to the gripper. After this marker transform has been computed and saved, the user can press the *Calibrate* button on the CoSTAR menu to calibrate a robot to a camera as long as the arm-mounted marker is visible via this pose estimate from CamOdoCal.

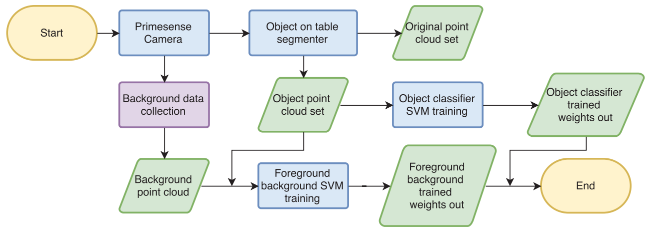
Start
Primesense Camera
Object on table segmenter
Original point cloud set
Background data collection
Object point cloud set
Object classifier SVM training
Object classifier trained weights out
Background point cloud
Foreground background SVM training
Foreground background trained weights out
End

Fig. 4: The training algorithm for the CoSTAR perception system. This procedure is a straightforward part of system setup and needs to be performed for each new object.

IV. Perception

The key challenge of integrating perception into an interactive programming environment is to ensure the usability of the system by exposing that perception in a way that makes sense to a human user and allows the user to build robust, comprehensible task plans. In addition, perception must produce high-level knowledge that allows users to create task plans that are robust to environmental variations such as the exact positions and orientations of particular parts or manipulation goals.

Fig. 2 shows data flow through our perception component and to the rest of our system. Raw RGB-D camera data is consumed by the AR marker tracker and by an object classification sub-component based on [12]. In our case we use a Primesense Carmine. Segmented point clouds are sent as raw input to an ObjRecRANSAC component [21] which performs pose estimation, creating a set of symbols describing individual object detections. These symbols are then sent to `Predicator` which produces predicates describing those symbols in terms useful to the end user. This behavior is exposed through a `DetectObjects` operation performed at specific known points in a task plan.

Our graphical programming system reasons over waypoints stored as 6-DOF coordinate frames that can come from one of three sources: (1) robot kinematics, (2) AR markers, and (3) the object detection and pose estimation pipeline. These coordinate frames are currently produced by ObjRecRANSAC [21] in the Pose Estimation step shown in Fig. 2.

Most symbols used by the CoSTAR system represent a position, either expressed in 6-DOF Cartesian space relative to a world frame or a joint space coordinate. The perception pipeline produces these symbols and associated predicates

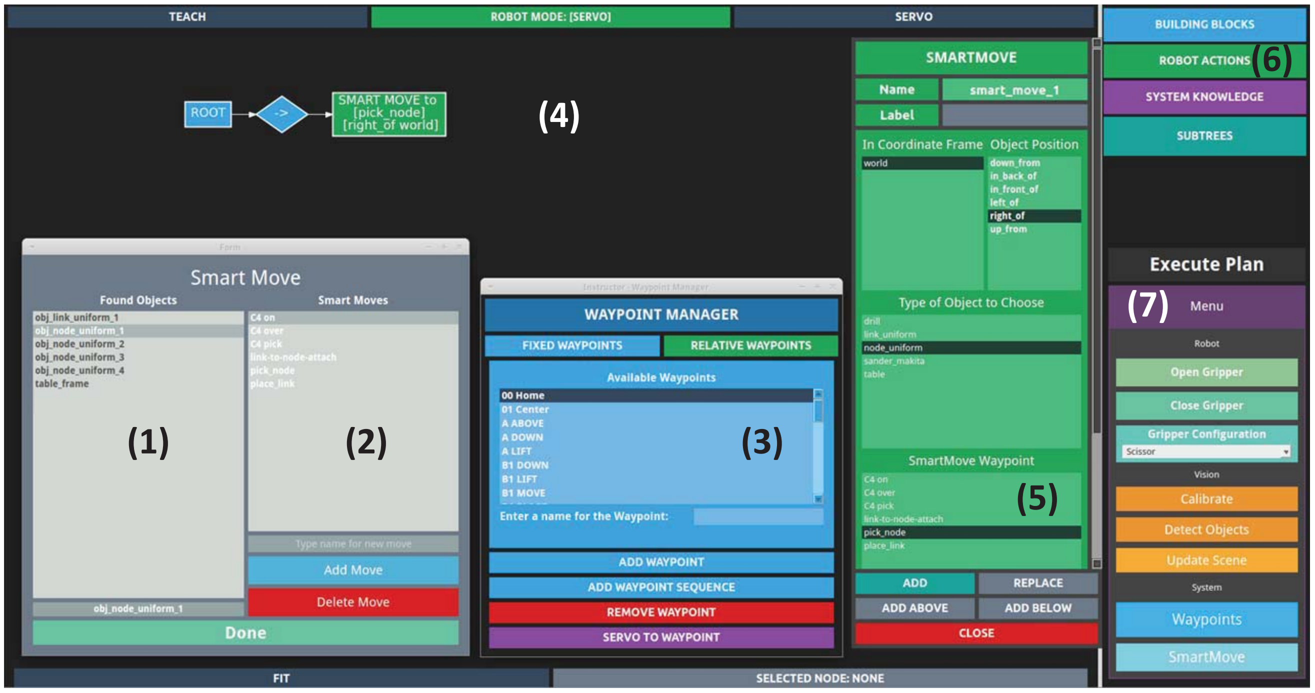

Fig. 3: Behavior Tree-based user interface. (1,2) Detected objects and associated SmartMoves. (3) Waypoint UI. (4) User's workspace containing the Behavior Tree. (5) SmartMove creation pane; Similar panes allow customization of other operations. (6) List of available operations. (7) Expanded menu containing simple operations the user can perform during plan development.

describing object class, and Predicator modules use these to assign values such as `LeftOf` and `RightOf` that describe relationships and other information pertaining to these frames.

A. *Object Pose Estimation Pipeline*

While the approach outlined above is effective at producing 6-DOF object pose estimates in cluttered environments, there are several challenges that must be handled specifically within the object pose estimation pipeline to ensure robust performance as part of a tool for authoring new task plans:

- It must be straightforward to adapt this system to new objects and new environments.
- Rotational ambiguities must be resolved consistently to allow for intuitive training of grasping approaches by end users.
- For some tasks, the identities of individual objects must be consistent across perception updates.
- Perceptual operations must be appropriately integrated with the CoSTAR behavior tree implementation to allow construction of complex programs.

Our object detection and pose estimation pipeline reduces sensor noise with a median filter before SVM segmentation and pose estimation. This produces accurate estimates of coordinate frames from the frame of reference of the camera. However, objects can contain pose ambiguities and symmetries that affect grasping angles. For example, an axis aligned solid cube can be rotated by 90 degree increments on any of three rotation axes and remain semantically equivalent. We define the function `setCanonicalOrientation()` which re-orients objects in the world frame based on a canonical orientation utilizing these symmetries, a prioritized axis ordering, and the particulars of the physical object model.

To retain object identities across multiple `DetectObjects` calls, we define the function `persistenceUpdate()`. This uniquely numbers each new detection instance and enters the frame position of each object into an R*-Tree [24], [25] spatial data structure, ideally bulk loaded via the R-Tree packing algorithm [26]. We utilize the Boost.Geometry Spatial Index R*-Tree [27] implementation. Upon each subsequent detection call a nearest neighbor lookup is performed and neighbors of the same class and within a certain threshold distance are assigned the names of previous detections. Objects new to the scene are assigned new names. See Alg. 1 for the complete procedure. This method has limitations, including velocity limits imposed by the perception update rate and ambiguities for objects with concavities.

When deploying our system in a new environment, we need three inputs: (1) an SVM for classifying point cloud data according to object type implemented as per [12], and (2) 3D models for each object class identified by the SVM for use with ObjRecRANSAC [21], and (3) symmetry

Algorithm 1 Maintaining persistent object identities across perception updates.

```
given R*Tree $R$, detected objects $\mathcal{O}_{detected}$, maximum distance $d_{max}$
function PERSISTENCEUPDATE($R, \mathcal{O}_{detected}, d_{max}$)
    $\mathcal{O}_{persistant} = \emptyset$
    for $o$ in $\mathcal{O}_{detected}$ do
        SETCANONICALORIENTATION($o$)  ▷ optional line
        $f_{nearestNeighbor}$ = NEARESTNEIGHBOR($o$)
        $f_{within}$ = WITHINDISTANCE($d_{max}$)[1]
        $f_{sameType}$ = SAMETYPE($o$)
        ▷ Query the R*Tree for the nearest neighbor $p$ with
        ▷ the same model type and within the max distance
        $n = R$.QUERY($f_{nearestNeigbor} \& f_{within} \& f_{sameType}$)
        if EXISTS($n$) then
            R.REMOVE($n$) ▷ enforce one match per object
        else
            SETUNIQUEID($o$)                    ▷ new object
        end if
        INSERT($\mathcal{O}_{persistant}, o$)
    end for
    ▷ Construct R*Tree with packing algorithm [26]
    return RTREE($\mathcal{O}_{persistant}$)
end function
```

information for these same objects. We outline a relatively quick data collection procedure that allows us to adapt to new environments and lighting conditions, shown in Fig. 4.

The training algorithm used for the state-of-the-art object segmentation approach in [12] requires a large amount of organized point clouds that provides multiple views of the object. Data collection is an intuitive process that can be performed by a non-expert user requiring only an RGB-D camera and a clear workspace. We use an AR tracking library[2] to specify the center of the workspace. Afterward, we do box segmentation with the center of table point cloud as the center of the box to isolate the table point cloud from its background. We also collect a large amount of "negative" data showing objects and surfaces that the system will not need to manipulate or will need to ignore, which in practice includes the robot and gripper. These point clouds are used in the SVM training in Fig. 4.

B. Perceptual Operations

We provide specific operations that allow users to integrate perception into a CoSTAR Behavior Tree. The `DetectObjects` operation updates the CoSTAR system's current representation of the world from RGB-D data, and the `SmartMove` operation performs queries that select symbols representing movement goals based on a list of required predicates.

The `DetectObjects` operation runs the perception algorithm described in Sec. IV-A and updates the robot's set of known waypoints and associated predicates. After a call to `DetectObjects`, the robot will have an updated list of the position, orientation, and object type of all objects it can identify in a scene. This is crucial for creation of *robust* task plans: it creates fixed points in the program at which the robot will update its knowledge of the world. Since users know when the `DetectObjects` operation is called, they can ensure the robot will have a non-occluded view of its workspace.

When authoring generalizable robot task plans, users need to be able to specify which objects they which to manipulate in an intelligent way. Our answer to this is the `SmartMove` operation. This operation queries `Predicator` to retrieve a list of all possible symbols matching a set of predicates. In our implementation, users can choose an object class and a geometry predicate, as shown in the user interface Fig. 3(6). For example, one might to find all objects of type `Part` that are `RightOf` a marker. The system will return the set of all symbols such that $p(s) \forall p \in p$, where P is the set of predicates to be matched. The SmartMove then uses object symmetry information for s to populate a list of possible poses that satisfy the given condition.

Then the operation selects and executes a feasible motion that will take it to one of these frames according to a cost function f. A cost function that minimizes joint space and Cartesian distance to the end effector provided reliable and believable behavior. This also allows us to intelligently choose between multiple object symmetries, for objects that can be grabbed or manipulated in several different ways. We use this capability to create structure assembly and collaborative tasks in Sec. V.

V. Evaluation

As discussed in the introduction, there are three characteristics that determine the power of a framework for authoring task plans:

- Capability: can the system in question perform a particular task?
- Usability: how easily can an end-user take an existing system and adapt it to a particular task?
- Robustness: will performances of a particular task plan be the same from one trial to the next, given reasonable environmental variation?

In this section we show these characteristics of our system by demonstrating the range of tasks that can be implemented with CoSTAR on each of our robots, and discuss task plan creation and repeatability. We published a YouTube playlist of experiments and use of our user interface. We assess our system through its applicability to a wide variety of tasks, and through a test of the repeatability of a structure assembly task using perception.

A. LBR iiwa Experiments

Our proposed system allowed us to quickly construct a wide variety of different tasks on-site at the Hannover Messe trade show. The LBR IIWA can carry a larger gripper from the UR5 (in this case a Robotiq S Model 3-finger gripper)

[1]Spatial relations based on the OGC Simple Feature Specification [28]

[2]AR Track Alvar: `http://wiki.ros.org/ar_track_alvar`

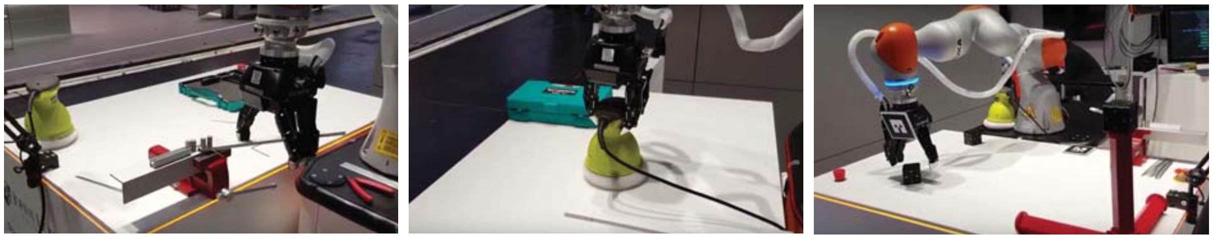

Fig. 5: Selected tasks implemented on the LBR iiwa using the CoSTAR system. From left to right: wire bending, polishing a surface, and collaborative structure assembly. For more videos see `https://www.youtube.com/playlist?list=PLCv90iHFljI3-VuVpUczGrNwvQOru3coZ`

and is mounted on a mobile cart. All tasks were constructed on site in roughly 30 minutes at the Hannover Messe trade show. Fig. 5 shows examples of these tasks.

Wire Bending: Our wire bending case study used a custom made wire bending jig. This use case demonstrates roughly the same set of capabilities as the system in [2]: we can quickly program a set of different capabilities, but this is all "blind": perception is not used because the wires are too small to detect and localize with our object detection system.

Sanding and Polishing: These tasks are very similar because they both use a similar tool that relies on Cartesian impedance movements. The robot must pick up the tool when it is available and move it along a known path to sand or polish some object until it is interrupted.

This task demonstrates the integration of arbitrary external hardware into the system with an additional component that exposes `ToolOn` and `ToolOff` operations. It also provides an example of preemption in behavior trees: we set up a Behavior Tree that uses a selector ("?") node, which means that it executes each child in order until one succeeds. This would either tick a subtree containing a "wait" gesture that moved the arm up and down to indicate that the robot was ready to pick up a tool, or it would tick a subtree containing a `RESET` node guarding a polishing procedure. The child of the `RESET` node would check to see if the tool was in position, and if this condition was false it would reset the child and return failure. The tree is shown in Fig. 6.

Collaborative Assembly: The robot picked up nodes from the right side of a table and waited. When a human made a particular gesture, the robot would hand the node to the human user or drop it and retry. Fig. 5(right) shows an example of this block task using the SmartMove operation described in Sec. IV-B.

B. UR5 Experiments

We created plans for three different tasks to demonstrate CoSTAR's ability to to construct multi-step tasks requiring precise manipulation using perception. We then completed the assembly task in three steps. First, the user programmed the UR5 to pick up any feasible node and lift it up. Then, the user used the existing task plan to pick up two nodes and move them from the right to the left side of the table. Finally, the user modified the node-manipulation program to pick up a new node and place it at the base of a structure.

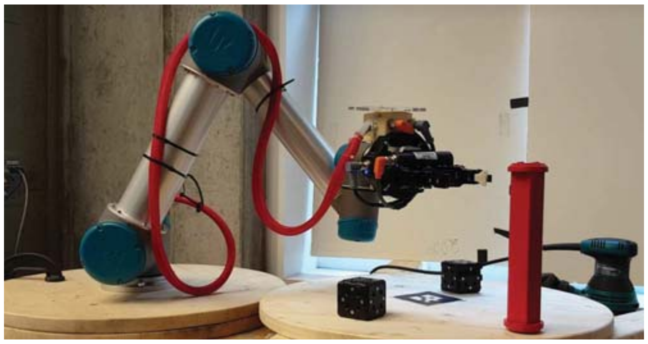

Fig. 7: UR5 performing assembly task.

After this the robot picked up a long connecting link, then finally a node to add to the task. The experimental setup is shown in Fig. 7.

We performed 10 trials of the final structure assembly task, completing it successfully in 10 out of 10 trials. We saw no perception failures in all 10 trials, although there was one experiment a notable pose inaccuracy due to sensor noise. In this case, the robot approached the block from a bad angle but was still able to successfully grasp it and place it because the position was corrected by the closure of the gripper. In general, we observed that barring sensor noise, the robot could complete the assembly with any set of objects for which it could find feasible grasps.

VI. Conclusion

We described a modular, cross-platform system for authoring robot task plans that is *capable* of capturing a wide variety of tasks, that is *usable* thanks to a powerful Behavior Tree-based user interface, and that is *robust* to changes thanks to abstract perception. Most importantly, our system grows more and more useful over time as users create task plans and develop new components. A user study to formally assess CoSTAR's usability is planned future work. Source code for basic CoSTAR components, including the Arm and Gripper components, perception, and controlling KUKA LBR iiwa hardware is available open source as a set of ROS packages[3].

[3]`https://github.com/cpaxton/costar_stack`

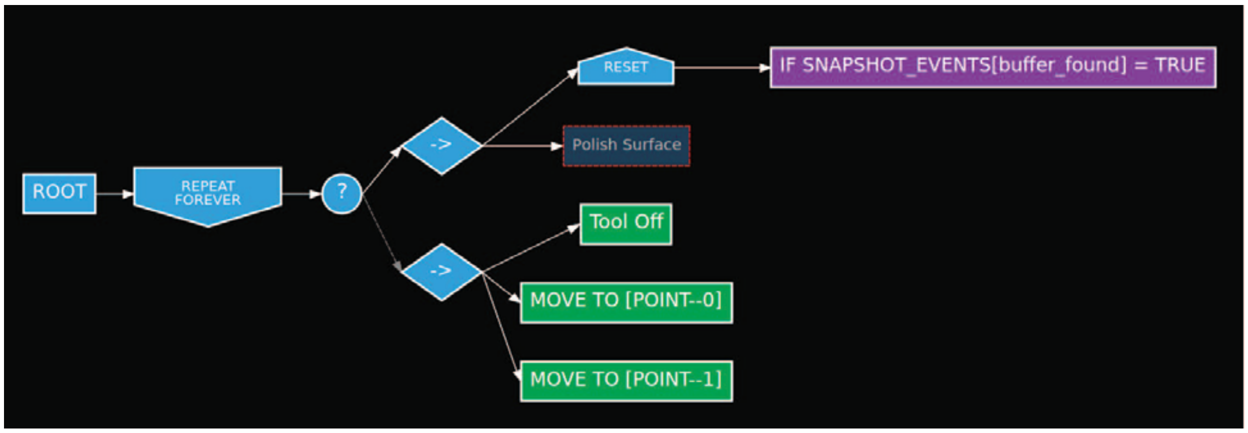

Fig. 6: Example of a complex task plan created for the polishing demo. The Behavior Tree includes iterator, sequence, and reset nodes. The subtree containing the actual polishing behavior has been collapsed for readability.

REFERENCES

[1] KUKA AG, "Call for participation KUKA Innovation Award 2016 Flexible Manufacturing Challenge sponsored by KUKA AG," https://www.kuka.com/-/media/kuka-corporate/documents/about-kuka/kuka_innovation_award_2016_call_for_participation.pdf.

[2] 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.

[3] C. Schou, J. S. Damgaard, S. Bøgh, and O. Madsen, "Human-robot interface for instructing industrial tasks using kinesthetic teaching," in *Robotics (ISR), 2013 44th International Symposium on*. IEEE, 2013, pp. 1–6.

[4] 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.

[5] 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.

[6] M. Beetz, D. Jain, L. Mosenlechner, M. Tenorth, L. Kunze, N. Blodow, and D. Pangercic, "Cognition-enabled autonomous robot control for the realization of home chore task intelligence," *Proceedings of the IEEE*, vol. 100, no. 8, pp. 2454–2471, 2012.

[7] S. Balakirsky, Z. Kootbally, C. Schlenoff, T. Kramer, and S. Gupta, "An industrial robotic knowledge representation for kit building applications," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 1365–1370.

[8] 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.

[9] J. Fasola and M. J. Matarić, "Interpreting instruction sequences in spatial language discourse with pragmatics towards natural human-robot interaction," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 2720–2727.

[10] Z. Kootbally, C. Schlenoff, C. Lawler, T. Kramer, and S. Gupta, "Towards robust assembly with knowledge representation for the planning domain definition language (PDDL)," *Robotics and Computer-Integrated Manufacturing*, vol. 33, pp. 42–55, 2015.

[11] 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.

[12] C. Li, J. Bohren, E. Carlson, and G. D. Hager, "Hierarchical semantic parsing for object pose estimation in densely cluttered scenes," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[13] S. Niekum, S. Chitta, A. G. Barto, B. Marthi, and S. Osentoski, "Incremental semantically grounded learning from demonstration." in *Robotics: Science and Systems*, vol. 9, 2013.

[14] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *arXiv preprint arXiv:1603.02199*, 2016.

[15] M. Tenorth and M. Beetz, "Knowrob: A knowledge processing infrastructure for cognition-enabled robots," *The International Journal of Robotics Research*, vol. 32, no. 5, pp. 566–590, 2013.

[16] 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.

[17] 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.

[18] 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.

[19] M. Colledanchise, A. Marzinotto, and P. Ogren, "Performance analysis of stochastic behavior trees," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 2014.

[20] D. Isla, "Halo 3-building a better battle," in *Game Developers Conference*, 2008.

[21] C. Papazov, S. Haddadin, S. Parusel, K. Krieger, and D. Burschka, "Rigid 3d geometry matching for grasping of known objects in cluttered scenes," *The International Journal of Robotics Research*, p. 0278364911436019, 2012.

[22] R. Smits, "KDL: Kinematics and Dynamics Library," http://www.orocos.org/kdl.

[23] L. Heng, B. Li, and M. Pollefeys, "Camodocal: Automatic intrinsic and extrinsic calibration of a rig with multiple generic cameras and odometry," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 1793–1800.

[24] A. Guttman, *R-trees: a dynamic index structure for spatial searching*. ACM, 1984, vol. 14, no. 2.

[25] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles," in *ACM SIGMOD Record*, vol. 19, no. 2. Acm, 1990, pp. 322–331.

[26] S. T. Leutenegger, M. A. Lopez, and J. Edgington, "STR: A simple and efficient algorithm for R-tree packing," in *Data Engineering, 1997. Proceedings. 13th International Conference on*. IEEE, 1997, pp. 497–506.

[27] A. Wulkiewicz and Others, "Boost.geometry spatial indexes," http://www.boost.org/libs/geometry.

[28] Open Geospatial Consortium, "OpenGIS implementation specification for geographic information - simple feature access - part 1: Common architecture," http://www.opengeospatial.org/standards/sfa.