A Framework for End-User Instruction of a Robot Assistant for Manufacturing

Kelleher R. Guerin, Colin Lea, Chris Paxton, Gregory D. Hager

***Abstract*— Small Manufacturing Entities (SMEs) have not incorporated robotic automation as readily as large companies due to rapidly changing product lines, complex and dexterous tasks, and the high cost of start-up. While recent low-cost robots such as the Universal Robots UR5 and Rethink Robotics Baxter are more economical and feature improved programming interfaces, based on our discussions with manufacturers further incorporation of robots into the manufacturing work flow is limited by the ability of these systems to generalize across tasks and handle environmental variation. Our goal is to create a system designed for small manufacturers that contains a set of capabilities useful for a wide range of tasks, is both powerful and easy to use, allows for perceptually grounded actions, and is able to accumulate, abstract, and reuse plans that have been taught. We present an extension to Behavior Trees that allows for representing the system capabilities of a robot as a set of generalizable operations that are exposed to an end-user for creating task plans. We implement this framework in *CoSTAR*, the Collaborative System for Task Automation and Recognition, and demonstrate its effectiveness with two case studies. We first perform a complex tool-based object manipulation task in a laboratory setting. We then show the deployment of our system in an SME where we automate a machine tending task that was not possible with current off the shelf robots.**

I. Introduction

There is increasing interest in the deployment of robotics in small manufacturing entities (SMEs) in order to enhance production efficiency and quality. However, SMEs present a number of unique challenges. First, SMEs must be able to rapidly adapt production lines for new products with small lot sizes. This requires robots to be easily repurposed to accomplish different roles in an ever-evolving workflow. Further, such repurposing should ideally be within the capabilities of a shop floor worker, and not require extensive training or specialized programming skills. Finally, systems have to adapt to unstructured environments where equipment, tools, and parts, most of which are not fixtured, may vary in position, orientation, or shape. While recent low-cost "collaborative" industrial robots have increased adoption of robotic automation in SMEs [1], many of these fundamental issues are still barriers that prevent SMEs from bringing robotic automation into their facilities.

To make these ideas more concrete, consider the examples shown in Figure 1A-C which we have taken from Marlin Steel, an SME that specializes in manufacturing metal wireform and sheet metal products. Although Marlin has five robots in its production facility, many manufacturing tasks continue to use hand tools (Figure 1A,B) for which there are no simple, flexible, cost-effective alternatives. However, tasks are highly repetitive and could, in principle, be automated if a robot could be easily trained or programmed to repeat the required motion. Tending available automation equipment such as the 2D planar wire bender shown in Figure 1C presents additional challenges due to the variability in part size and shape, and requires accounting for the changing positions of mobile racks and the variable timing of the aging wire bender. As a result, Marlin uses a human worker to collect the completed parts as this is more cost effective than attempting to program an industrial robot to perform this "simple" transfer task. As we will further describe below, our system, CoSTAR, was successfully deployed in roughly 30 minutes to perform this task as shown in Figure 1D.

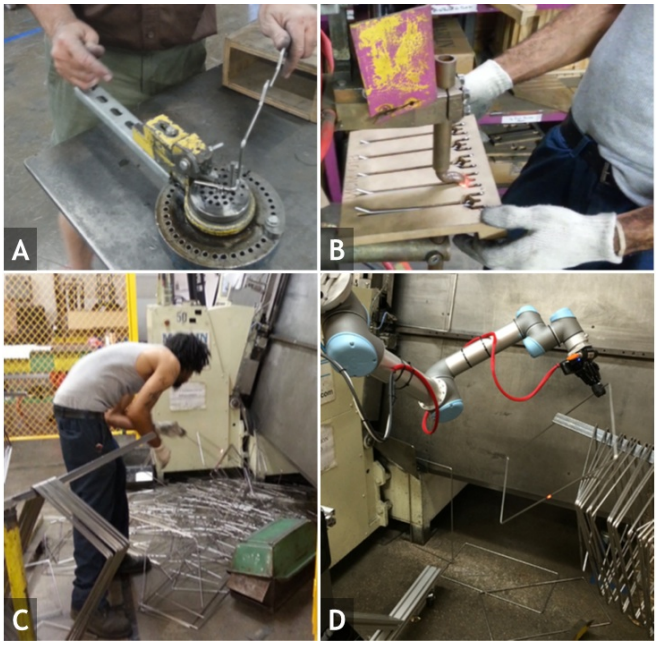

Fig. 1: Typical tasks at an SME specializing in metal products. (A) shows a hand bending task, (B) shows a spot welding task, (C) shows a 2D wire bender tending task, and (D) shows the system presented in this paper accomplishing the same task as the human worker in (C).

Our goal is to develop a practical framework for deployment of robot assistance in an industrial setting that addresses the limitations outlined above. More specifically, our approach is to develop a system for instructing a robot

*This material is based upon work supported by the National Science Foundation under Grant No. NRI-1227277. Authors can be reached by email at: {kguerin2, clea1, cpaxton3}@jhu.edu and hager@cs.jhu.edu.

All authors are with the Dept. of Computer Science, Johns Hopkins University, Baltimore, MD, USA.

assistant that has the following attributes:

1) The system contains a nominal set of capabilities that are consistent with the tools, processes and working environment of a typical small manufacturer.
2) The system can combine elements of traditional programming and direct demonstration in a flexible, easy-to-use, and principled manner to extend capabilities.
3) The system is perceptually grounded so that task performance is robust to modest changes in the environment and task descriptions can be abstracted with respect to the objects involved.
4) The system is able to accumulate, abstract, and reuse plans that it has been taught akin to how an assistant gains skill with instruction and experience.

In the remainder of this paper we describe the conceptual framework for CoSTAR, a system for collaborative robotics for manufacturing. We describe the core execution framework we have developed for CoSTAR and emphasize the use of Behavior Trees as the core task representation. We describe how Behavior Trees are connected with a set of robot capabilities, how they are perceptually grounded, and how abstract task descriptions result. Additionally, using Behavior Trees as a task specification allows transparent analysis and generalization of structure, composition, and classification of execution traces relative to a dictionary of known tasks. We detail the implementation of CoSTAR and describe its application to processes like kitting and material transfer in industrial manufacturing and describe the results of a pilot deployment at Marlin Steel.

II. Related Work

There is a large space of work on improving interaction with robots for industrial applications, as well as task modeling and knowledge representation for robots.

A. Robot Assistants and Collaboration

There have been several efforts for creating collaborative robotic assistants for industrial tasks. The robot assistant in the rob@Work [2] project could be programmed by the user with specific actions. The PowerMate project [3] provided human-robot collaboration based on the force exerted by the user. More recent work by Hawkins et al. performed assistance by recognizing which part of the task the user is executing [4]. We take a hybrid approach to collaboration by explicitly defining a task plan for the robot and providing perceptually grounded events the system can detect as input for certain actions. Finally, robot safety has been addressed by others (e.g. [5], [6]); we specifically do not address the safety of collaborative industrial robots.

B. Robot Task Definition

Numerous systems exist for building complex robot task plans based on sensor data. SMACH is a ROS-based framework for building robot task models as hierarchical concurrent state machines [7]. ROSCo built upon SMACH to provide an easy-to-use system for defining perceptually grounded task plans in a home environment [8]. Other work by Pedersen et al. used a gesture-based interface to create a task plan for a mobile robot in an industrial setting [9].

We build upon Behavior Trees as an alternative to these approaches. Behavior Trees are gaining popularity as a model for creating generalizable robot programs [10], [11]. While initially designed for large industrial processes, more recently they have been used to implement character AI in video games [12]. Behavior Trees have also been shown to provide a more modular, adaptable representation of a robotic task than traditional models like Finite State Machines [13]. In this paper we describe how low-level robot capabilities, perceptual grounding, and generalizable task descriptions results using Behavior Tree specification.

C. Knowledge Representation

Research in task specifications in terms of symbolic properties of the world dates back to STRIPS [14]. More recently, KnowRob was able to parse human-readable task plans from web sources and turn them into task plans for a PR2 using OWL ontologies [15]. Work by Balakirsky et al. used a robot ontology specified in PDDL together with a high level Canonical Robot Command Language converted into low level commands to perform a kitting task [16]. Similarly Huckaby et al. described the SysML description language for specifying robot tasks [17]. Other work by Dantam et al. parsed sensor data into symbolic tokens which could be meaningfully processed by a robot to allow it to safely interact with a human when playing a game of chess [18]. These systems all assume a detailed ontology exists for the task, and therefore are not conducive to creating new tasks in new domains on the fly. Our system does not have a preconceived task model or ontology; rather, we rely on the human's domain specific knowledge to define the task, allowing for flexibility to new tasks and domains.

III. A Framework for Robot Task Specification

Our proposed framework aims at representing tasks in a way that provides easy task manipulation, visualization, and analysis. It is designed with the mindset that a robot systems programmer will create a set of capabilities and an end-user will interactively create a task plan with the exposed functionality. First we define the characteristics of system capabilities. We then describe how these capabilities ground a variant of Behavior Trees that supports task abstraction which enables easy generalization to new situations. Throughout the rest of this section we reference the following transport task: detecting a cup on a table and placing it in a waste bin.

A. System Capabilities

The core capabilities of any robotic system are ultimately derived from a set of software and hardware components. Typically these components are grouped to provide a concrete functionality (i.e. perform force-based impedance control, perform object detection, or control a gripper). We define a *system capability* as a three-tuple $\Omega = \langle \mathbf{O}, \gamma, \phi \rangle$ where $\mathbf{O}$ is a set of exposed operations, γ is a set of

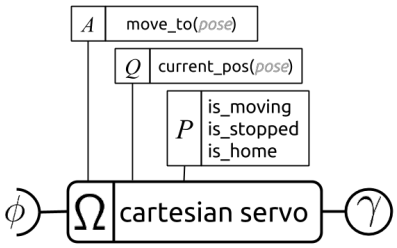

Fig. 2: Diagram of a single capability for Cartesian movement, exposing an action for movement, a query for the current pose, and three predicates for testing whether the system is moving, stopped or in the home position.

continuous input to the capability, and ϕ is a set of continuous output from the capability. γ and ϕ are included to place constraints on the information interfaces (i.e. sensor signals and hardware) a system must provide to implement the capability. For example, Figure 2 shows the capability Ω_{cart_servo} for commanding Cartesian motion to a specified pose. In this case, ϕ would be encoder readings from the robot, γ would be commanded joint torques, and $\mathbf{O} = \{move\_to, current\_pose, is\_moving, is\_stopped, is\_home\}$.

Capabilities produce a set of outputs γ and can only be used when the required inputs ϕ are available. Capabilities can be composed to build more complex systems: for example, an object detection capability might be composed with an object tracker for real-time recognition. For more information on composition of capabilities, see [19].

Operations $\mathbf{O}$ are the set of high-level functions exposed by a system to a user for creating task plans. A capability Ω may expose multiple instances of three types of operations: actions A, queries Q, and predicates P. An action operates on a set of specified parameters and when evaluated produces an output in γ that commands the robot. Example actions include moving to a Cartesian position, opening a gripper and exerting a force. A query requests some data from the exposing capability and stores the value to a parameter. From the transport example, a query could retrieve the Cartesian pose of the cup. In the wire bending example a query could determine the state of a manufacturing machine. A predicate returns only a success value indicating the truth of some logical statement about the world. For example, if the user wants to ensure that the robot is grasping an object obj then they would check if the predicates $P_{is\_closed}(gripper)$ and $P_{touching}(obj, gripper)$ are `True`.

B. Behavior Trees

Behavior Trees provide a structured specification for robot tasks. Here we briefly describe the formulation. For a more detailed explanation see [10], [11]. A Behavior Tree $T = \{R, \mathbf{N}, \mathbf{L}, \mathbf{E}\}$ is an ordered tree with root R, leaves $\mathbf{N}$, internal nodes $\mathbf{L}$, and edges $\mathbf{E}$. The leaf nodes, $\mathbf{N}$, are evaluated at runtime, and the internal logical nodes, $\mathbf{L}$, provide a control-flow structure to determine how and in what order the operation nodes are evaluated. The value of a logic node $L \in \mathbf{L}$ with some number of children $l_1, \ldots, l_n$ is defined by evaluating its children according to one of four policies:

- *Parallel* nodes evaluate all children simultaneously and return success only if all children return `SUCCESS`. In the transport example the cup and the waste bin can be detected simultaneously.
- *Sequence* nodes evaluate each child in order only if the previous child evaluates to `SUCCESS`, and returns `SUCCESS` once all children have returned successfully. In the transport example the sequence of actions starting with moving to the cup would be executed in this manner.
- *Selector* nodes evaluate each child in order until any child returns success, at which point the selector returns `SUCCESS`. In the transport example a selector could be used to "catch" a failure, like in detecting the cup, and instead command the robot to wait.
- *Decorator* nodes only have one child and are used to modify the return value of its child. An *iterator* decorator resets its child upon receiving a `SUCCESS`. This is useful for repeatedly evaluating a sub-tree as it succeeds. An *enforce* decorator resets its child upon receiving a `FAIL`, which is useful when the success of a node is required in order to continue.

Evaluation of a tree begins at the root node, R, and occurs periodically with frequency δ. This recursive evaluation is called a "tick". Once all nodes in the tree are evaluated, the root returns the value of its child. In general if the tick reaches a node that has already returned either `SUCCESS` or `FAIL`, that node will not be evaluated again. For all logical nodes, when the evaluation of a child returns `RUNNING`, meaning the child is running some processes that takes longer than δ, the parent will return `RUNNING`. When a child node has returned `SUCCESS`, it will not be re-evaluated if ticked again. In this case it is often necessary to reset a node explicitly, if it needs to be evaluated with every tick propagation. Following the behavior tree formulation in [10], this reset event can be initialized by a root or decorator node (such as the "enforce" node above) and causes the entire sub-tree under the root or decorator node to be reset.

A Behavior Tree typically has two types of leaf nodes $N \in \mathbf{N}$, "Actions" and "Conditions". When an action is evaluated, it invokes some discrete physical process or event in the world. An action return `SUCCESS` if the action finishes successfully, `FAILURE` if the action fails, and `RUNNING` if the action has not completed. Condition nodes serve to guard or enable the evaluation of sibling nodes. Evaluating a condition typically performs some test against a fixed value. If the test is `TRUE`, the condition will return `SUCCESS`, otherwise the condition returns in failure.

C. Task Specification

Ultimately, we would like the robot to perform some task by leveraging its capabilities from Section III-A, by using its exposed action operations along with information made

available through queries and predicates. We therefore formulate a *task specification* as a set of exposed operations that are completed by the robot to achieve some task. An integral component of this task specification is the logical structure that determines in what order and under what conditions each operation takes place. Therefore, a task specification at its core is a Behavior Tree where the leaf elements of the tree are modified to expose capability operations. Formally, a task specification is the tuple $\mathcal{T} =< T, \Omega, \mathbf{X} >$, where T is a Behavior Tree, Ω is the set of capabilities required by the task, and $\mathbf{X}$ is a set of parameters required to instantiate the operations exposed by Ω.

In a task specification, the leaf nodes of T correspond to exposed operations from the capabilities Ω. A behaivor tree action corresponds to a capability action operation, and follows the same evaluation pattern as described in Section III-B. When this action operation is evaluated, it will send some command, such as "move" to the robot. Queries are another operation that now acts as a behavior tree leaf node. When evaluated, a query will expose some piece of data to the rest of the nodes in the tree. This data is held in a named parameter that other nodes in the tree can use. This is similar to what is sometimes called a blackboard (i.e. [20]). For instance in the transfer task above a query could expose the position of the cup as a parameter `CUP_POSE` which can be used by other operations. Capability predicates are exposed as behavior tree conditions in a task specification. Rather then testing some value as described above, a task specification condition tests if a capability predicate is true. For example in the transfer task above, assuming a capability has the predicate `HOLDING`, a condition can be used to check if the robot is holding a cup.

A *task plan* is a task specification where all parameters required by the Ω have been defined. For example, a task specification for the transfer task above would include operations to find an object, move to its location, grasp that object, and release. In a defined task plan the object would be set to value `CUP` and the release location would be set to the pose of the waste bin. The instantiation of parameters in a task plan can be done by the user for "fixed" parameters or can be generated by queries. The graphical user interface used to instantiate fixed parameters is discussed in Section IV-A.

Figure 3 shows the transfer task. In the diagram, logical nodes are symbolized in white: root "(/)", parallel "|A|", selector "(?)" and sequence "$-$ $>$". The colored operation nodes have a field for parameters such as `CUP_POSE` and `CUP`. In the task the parameters `CUP` and `BIN` represent object detection models and the parameters `CUP_POSE` and `BIN_POSE` represent Cartesian poses.

This plan operates by detecting a cup and bin object and then checks a condition to see if the cup is found. If so, it queries the location of the cup and bin and store those poses in parameters `CUP_POSE` and `BIN_POSE` respectively. The system then performs the movement and grasp sequence starting with action `MOVE_TO[CUP_POSE]` and ending with `MOVE_TO[BIN_POSE]` where the system moves to

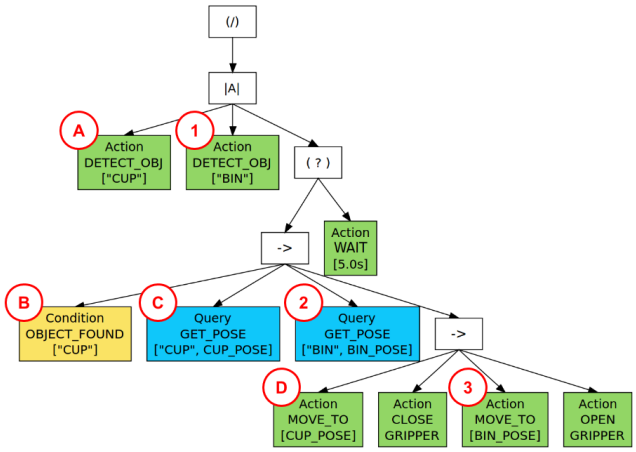

Fig. 3: A parameterized task plan for finding a cup and dropping it in a waste bin. If the cup is not found or the query for the pose of the cup or bin fails then the system will default to a sleep action. The flow of information through the tree is also illustrated: (A) detection of the cup, (B) a condition about its successful detection, (C) a query of its pose, and (D) the use of that pose in a movement command. Similarly (1) detection of the bin, (2) a query for the bin pose and (3) a movement to the bin. (Note: in this figure, parameters in quote such as `"CUP"` are fixed parameters defined by the user, where as parameters not in quotes are exposed by query.

the pose of the cup, grasps it, and moves to the pose of the bin where it is dropped. Should any of the conditions or queries fail the system will fallback to the `WAIT` action. The assumption is this tree is being evaluated until it returns SUCCESS. Therefore if an object is not found initially but appears in a later evaluation of the tree, `OBJECT_FOUND` will return SUCCESS and the tree will continue.

By changing parameter assignments we can easily re-parameterize operations to adapt them to different tasks. For example the task plan in Figure 3, which is instantiated for locating a cup and dropping it in a bin, could easily be re-instantiated for another object such as a plate by changing the parameter assignment of `CUP` to a model `PLATE`.

IV. CoSTAR System

We implemented our framework in a new system called CoSTAR, the Collaborative System for Task Automation and Recognition, as shown in Figure 4. CoSTAR consists of a set of system capabilities for performing perception (Perceptor), inferring complex object relationships (Predicator) and human interaction (Collaborator). These are combined with a task plan and a task definition environment (Instructor). First we discuss Instructor, a user-friendly task definition environment for defining task plans and deploying them on a robot. Then we highlight the capabilities implemented in our system that are necessary for performing real world tasks. All of these components are implemented using the Robot

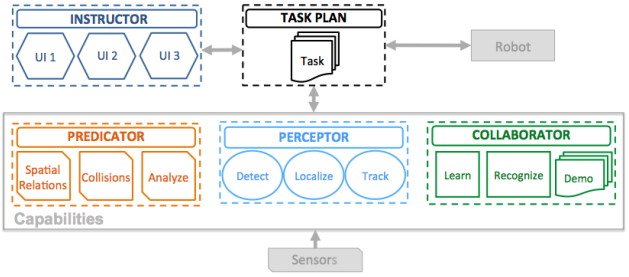

Fig. 4: System overview for CoSTAR: the Collaborative System for Task Automation and Recognition. CoSTAR is comprised of a set of system capabilities, a task plan, and user interface for creating task plans.

Operating System (ROS) [21], and work cross-platform with a variety of tools and robots.

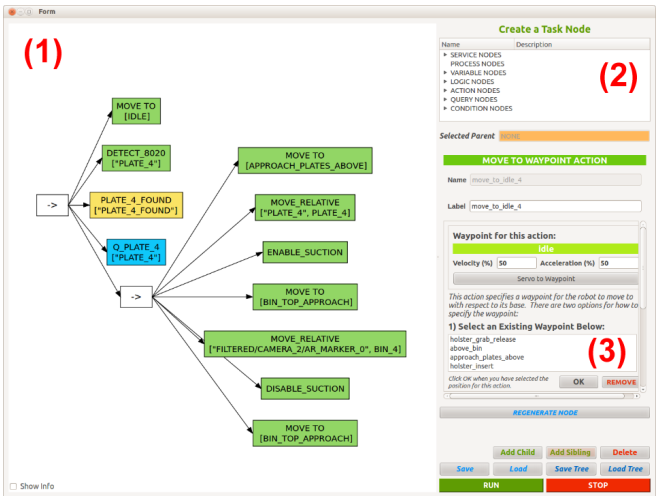

Fig. 5: The Instructor user interface for creating task plans and executing them on the robotic system. The canvas includes areas where the user can (1) create a task plan, (2) select capability operations and logical internal nodes, and (3) specify parameters for the plan operations. The task plan shown here is for transferring a metal plate.

A. Instructor

Instructor consists of a graphical user interface for building a robot task plan. The Instructor graphical user interface (GUI) is shown in Figure 5. The GUI contains a canvas displaying the currently composed Behavior Tree, as well as a sidebar for selecting different logical and operation nodes to compose in that tree. Instructor exposes operations from the capabilities available to the user. This adds modularity to our system because operations are automatically added for the end-user as new capabilities are added to the system.

Some system capabilities provide a GUI for defining the specific parameters required for configuration of their defined operations. For instance, an object detection operation includes a GUI for specifying a class of object to be detected. Parameters can also be specified by kinesthetic teaching. For

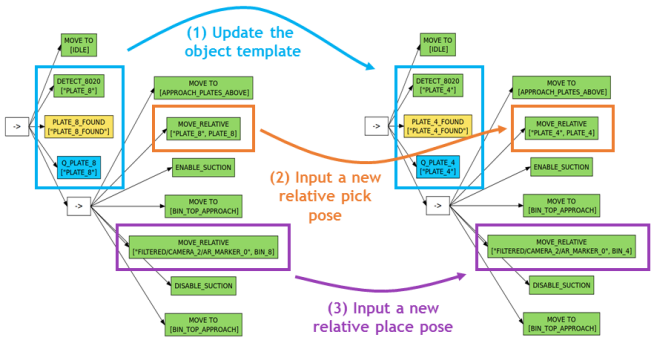

Fig. 6: A sub-tree for transporting an 8-holed metal plate can be easily adapted for a different 4-holed plate by: (1) specifying a new object model for the 4-hole plate, (2) demonstrating a new pick position relative to the 4-hole plate, and (3) demonstrating a new placement position relative to the bin.

example the user can add a waypoint manually by moving the robot to a specific location for navigation.

The user can save a sub-tree for reuse in other task plans. This sub-tree may be saved with or without the specified operation parameters. With a set of saved subtrees the user can create a complex task plan composed of several smaller subtrees. This is shown in Figure 7 for the example task performed in Section V-A. The initial tree (A) contains nodes for recognizing that a bin has been placed in the workspace and returning the bin to a specified location. (B) adds a sub-tree to unholster a suction gripper, (C) adds a sub-tree for transporting a metal plate to a bin, and (D) adds a sub-tree for finally holstering the suction gripper. In addition these sub-trees can be reparameterized so that the same sub-tree can be used for multiple actions. This is shown in Figure 6. When sub-trees share variable names it is up to the user to make sure there are no collisions.

B. CoSTAR System Capabilities

The following capabilities have been implemented to facilitate use of CoSTAR on real world tasks. Other capabilities for controlling our robot, such as velocity control, are also implemented but not discussed. The different capabilities implemented in CoSTAR are shown in Figure 4.

Two object recognition capabilities are implemented to detect and localize objects in the environment. We use an accurate model-based object recognition system similar to [22]. We combine this with a simple surface segmentation based object tracker. Other vision capabilities were implemented specifically to classify metal plates of varying sizes for one of our tasks described in Section V. Collectively, these capabilities are referred to as Perceptor in Figure 4.

A capability for occupancy detection was implemented that returns a predicate `is_occupied` when there are entities in a selected 3D volume and `is_empty` when it contains no entities. This region is specified as a sphere in world coordinates by the user through a graphical interface during the creation of the task plan. This module can be used

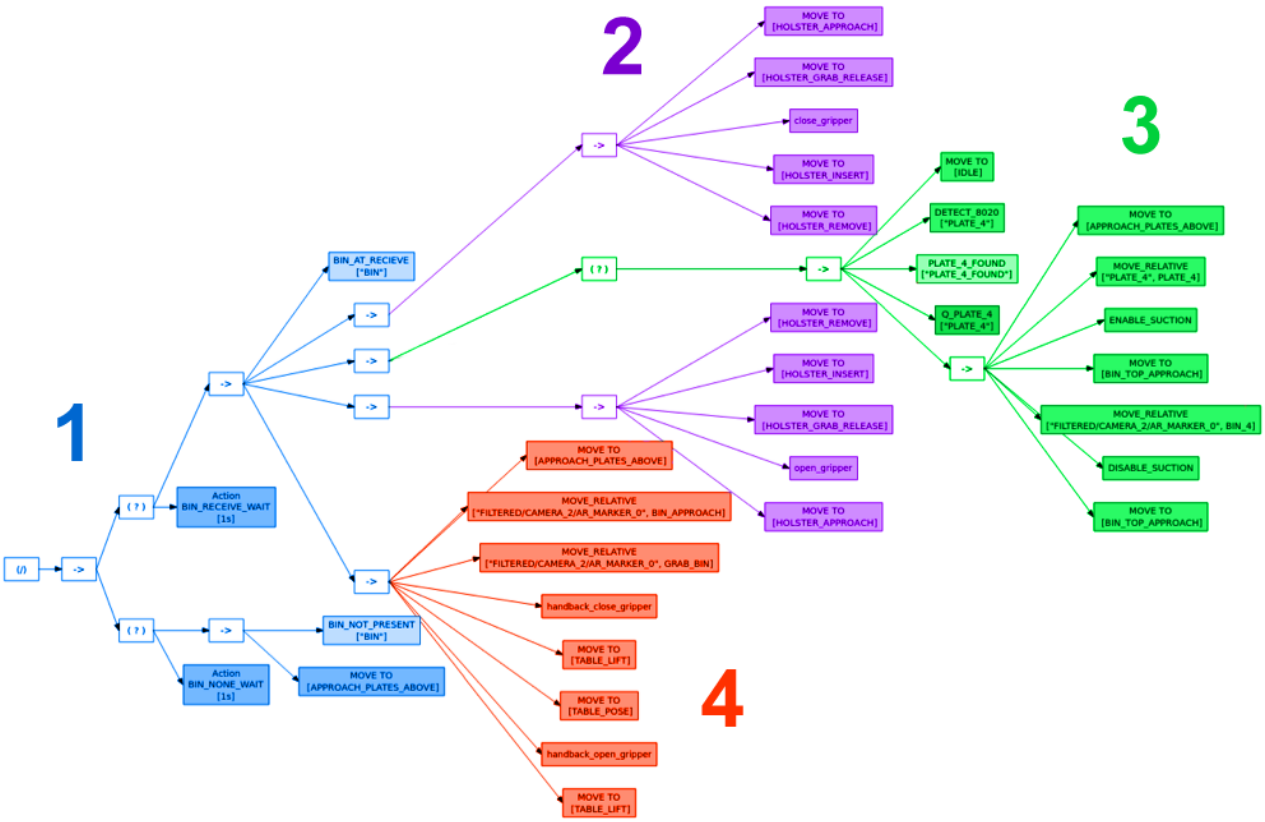

Fig. 7: The task plan for kitting in Section V-A, showing the composition of small sub-plans into a more complex plan. (1) shows the initial task plan for responding to the handoff of a bin in the workspace, (2) shows the addition of sub-plans for unholstering and holstering the suction tool, (3) shows the addition of a pick and place sequence for a plate, and (4) shows a sub-plan for returning the bin to the user at the end of the task.

to detect changes in environment state such as the movement of a machine in a manufacturing task.

We use an AR tracking capability[1], to track objects that we cannot accurately recognize using computer vision. This capability is also used to calibrate the cameras on our robot based on the relative locations of known markers in all cameras.

A spatial relationship capability computes predicates based on the pose of known objects in the environment. These predicates include `left_of`, `right_of`, `in_front_of`, `behind_from`, `up_from`, and `down_from` given in various frames of reference. The position for each object is aggregated over time and is used to compute other predicates indicating whether or not objects are moving relative to each other. We use the MoveIt ROS interface to compute whether or not objects are in contact or are near collision, based on known object meshes and position information. These capabilities are collectively referred to as Predicator.

Finally, CoSTAR includes a collaboration capability (Collaborator) that facilitates human-robot interaction for complex tasks. Currently collaboration is limited to the detection of events that a human triggers in the world, like moving an object into place. Future work will incorporate gesture recognition and fine grained activity analysis to enable more complex interactions.

[1]ALVAR: http://virtual.vtt.fi/virtual/proj2/multimedia/alvar.html

V. Case Studies

We present two case studies using a Universal Robots UR5 robot. The first is in our laboratory and another is at an SME. There are two RGBD sensors to monitor the entire environment: one on the "shoulder", above the robot stand, and one mounted on the "wrist", next to the robot end effector.

A. Laboratory Experiments

We identified small lot spot welding and kitting as two common industrial tasks that require complex capabilities difficult for an off-the-shelf industrial robot. Spot welding is shown in Figure 1B and kitting is examined in other works such as [16]. In the spot welding task, the user inserts parts into a fixture to first perform quality assurance and then to fix them into place for welding. The user then takes the parts over to a spot welder and presses a foot petal to weld in each of the five different positions. This task requires a human because quality assurance using the fixture is a very difficult perception problem and is impractical to automate using current technology. To complete this task using CoSTAR, we would develop a collaborative task plan where the human

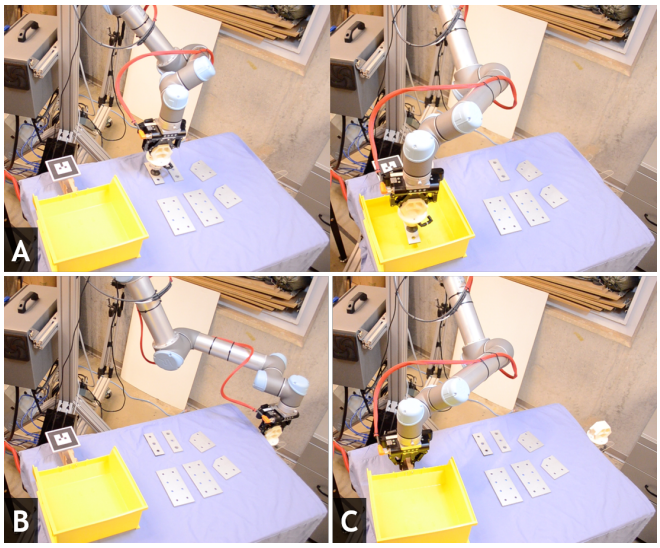

Fig. 8: Several sub tasks composed into the complete task plan. (A) shows `Transfer` of a metal plate, (B) shows the robot in position to `Unholster` the suction gripper, (C) shows the grasping of the bin for `Return` to the final location.

would perform the quality assurance and fixturing and the robot would perform the time consuming spot welding.

Kitting is another common task performed by SMEs in which a human worker finds one of each part and moves them to a specific location. This task is very repetitive and requires accurate object detection and localization.

We demonstrate the application of CoSTAR to these types of tasks with an example performed in the laboratory. We use CoSTAR to teach the robot to create a kit of three metal plates of varying sizes by first putting one of each part in a bin that the robot receives from a worker. In this case study the robot waits for a human to deliver a bin and then returns the bin with a set of plates at a different location at the end of the task. This exemplifies the type of human-robot collaboration necessary for the aforementioned spot welding and kitting tasks.

We start by creating five simple task specifications rather than starting with one specialized task plan:

- `Unholster`: the robot retrieves a suction gripper
- `Holster`: the robot returns the suction gripper to its holster
- `Transfer`: the robot performs a simple pick-and-place action. The robot locates a metal plate and moves it from a position on a table to a specified destination in a box
- `Handoff`: the robot waits for the worker to put the bin on the table and then acknowledges that it has found the bin
- `Return`: the robot picks up the bin and puts it in a new location

We then compose a collaborative task plan as shown in Figure 7. The robot first waits for a `Handoff` from the user, uses `Unholster` to pick up a suction gripper, then `Transfer` to move one of each of three types of metal plates to a bin. It uses `Holster` to return the suction gripper to its designated storage location, so that its gripper is free, and then uses `Return` to move the bin to a new location.

Figure 8 shows one configuration of the experiment during execution of the `Transfer`, `Unholster`, and `Return` sub-tasks for the 3-hole metal plate. This task plan is robust to changes in position and orientation for the objects and bin. In addition, in the `Transfer` portion of the task the robot will grab any of the three parts that is available. This shows CoSTAR's ability to adapt to more complex task structures; CoSTAR tasks do not need to be a simple sequence of action primitives.

B. Manufacturing Experiments

To test CoSTAR in a real world situation we deployed the system at Marlin Steel Wire, an SME specializing in wireform and sheet metal products. As with many SMEs, their typical lot sizes are 100 to 1000 parts. They require quick turn-around times and frequent (sub-daily) changeover between products. Specifically, our system performed machine tending with an aging 2D wire bender. This wire bender produces 2D wireforms at irregular intervals and lacks a communication port to coordinate with the robot. A human worker collects the wireforms produced by the bender from the ground and puts them on a mobile cart.

We first positioned the robot near the wire bender and then calibrated the cameras using AR markers placed on the wire bender and the mobile rack. We parameterized an occupancy operation to detect the state of the machine by looking at the moving welding arm on the machine. The event where the machine moves its arm to release a part is recognized so the robot can then grab the part. At that point the UR5 picks up the wireform at the exit position from the machine and transfers it to the rack. The procedure the robot follows is shown in Figure 9. The entire setup process took two hours, including setting up the robot with a stand and two RGBD sensors. Creating a parameterized task plan took roughly 30 minutes, and no code was written on-site.

Trial	Bad Weld	Bad Grasp	System ESTOP
1	1	2	6
2	1	3	7

TABLE I: The causes and number of failures in 2D wire bender tending task for two 50-part trial runs.

We conducted two experiments where in each run the machine made 50 parts and the robot placed each one on a rack. Despite being the first real world deployment of our system, we achieved 82 successful "catches" out of 100 parts produced. Table I shows the causes and numbers of failures. Most of the failures were caused by system ESTOP: when there was either a bad weld or a bad grasp the robot would need to be stopped and reset. Even though this only happened a small number of times it caused a large number of dropped wireforms because the bender kept producing parts. These

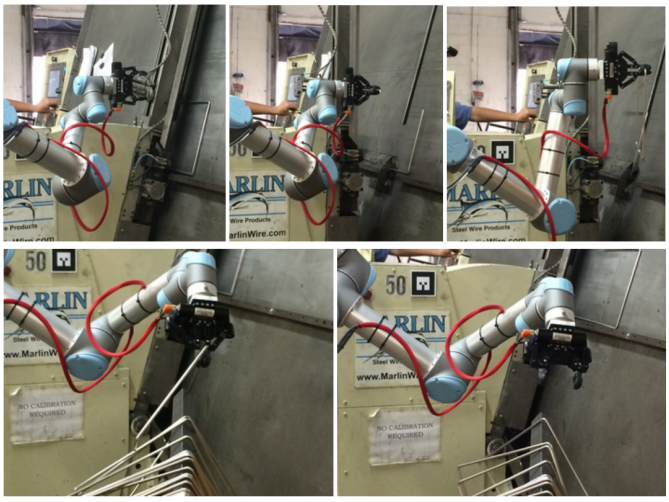

Fig. 9: A sequence of five images showing the successful execution of the 2D wire bender tending task.

errors could be prevented through better error handling in the CoSTAR task plan, which would have brought our success rate up to 93%.

It is simple to adapt this process to a new part: the user can simply change the waypoint used to grasp the wireform. The task could also be easily integrated into a more complex process as demonstrated in Section V.

VI. Conclusion

We have described a framework for end-user instruction of industrial robot assistants that enables complex, event-driven automated behaviors. Then, we outlined how this approach can be used to create generalizable task models that can be composed to accomplish more complex tasks. This framework was implemented in a system called CoSTAR, which has been deployed in both laboratory experiments and real industrial tasks at an SME. The capabilities of this system correspond to tools and processes crucial to common tasks in a manufacturing environment. Our case studies demonstrated how CoSTAR can quickly create programs that perform common industrial tasks in a way that is robust to changes in that environment.

In the future, we will perform a user study showing the ease of use of the proposed system. We will expand the capabilities of the CoSTAR system focused on human-robot collaboration to handle more complex interactions. In addition we will work on formalizing how capabilities are abstracted across different robot platforms and continue work on how information can be represented in Behavior Trees. Finally, we plan on developing a set of tools to evaluate a collaborative task to determine where best to use a robot assist in manufacturing.

References

[1] P. Waurzyniak, "They're here: New collaborative robots lend a helping hand," Jun. 2013. [Online]. Available: http://www.sme.org/MEMagazine/Article.aspx?id=73500

[2] E. Helms, R. D. Schraft, and M. Hagele, "rob@ work: Robot assistant in industrial environments," in Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on. IEEE, 2002, pp. 399–404.

[3] R. D. Schraft, C. Meyer, C. Parlitz, and E. Helms, "Powermate–a safe and intuitive robot assistant for handling and assembly tasks," in Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on. IEEE, 2005, pp. 4074–4079.

[4] K. P. Hawkins, S. Bansal, N. N. Vo, and A. F. Bobick, "Anticipating human actions for collaboration in the presence of task and sensor uncertainty," in Robotics and Automation (ICRA), 2014 IEEE International Conference on, 2014.

[5] A. Vick and J. Krüger, "Safe physical human-robot interaction through sensorless external force estimation for industrial robots," in HCI International 2013-Posters Extended Abstracts. Springer, 2013, pp. 616–620.

[6] F. Flacco and A. De Luca, "Safe physical human-robot collaboration." in IROS, 2013, p. 2072.

[7] J. Bohren and S. Cousins, "The smach high-level executive [ros news]," Robotics & Automation Magazine, IEEE, vol. 17, no. 4, pp. 18–20, 2010.

[8] 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.

[9] M. R. Pedersen, D. L. Herzog, and V. Krüger, "Intuitive skill-level programming of industrial handling tasks on a mobile manipulator," in Intelligent Robots and Systems (IROS), 2014 IEEE Conference on. IEEE, 2014.

[10] 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.

[11] M. Colledanchise and P. Ogren, "How behavior trees modularize robustness and safety in hybrid systems," in Intelligent Robots and Systems (IROS), 2014.

[12] C.-U. Lim, R. Baumgarten, and S. Colton, "Evolving behaviour trees for the commercial game defcon," in Applications of Evolutionary Computation. Springer, 2010, pp. 100–110.

[13] A. Shoulson, F. M. Garcia, M. Jones, R. Mead, and N. I. Badler, "Parameterizing behavior trees," in Motion in Games. Springer, 2011, pp. 144–155.

[14] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," Artificial intelligence, vol. 2, no. 3, pp. 189–208, 1972.

[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] S. Balakirsky, Z. Kootbally, T. Kramer, A. Pietromartire, C. Schlenoff, and S. Gupta, "Knowledge driven robotics for kitting applications," Robotics and Autonomous Systems, vol. 61, no. 11, pp. 1205–1214, 2013.

[17] J. Huckaby and H. Christensen, "Modeling robot assembly tasks in manufacturing using sysml," in ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of. VDE, 2014, pp. 1–7.

[18] 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.

[19] K. Guerin, S. Reidel, J. Bohren, and G. D. Hager, "Adjutant: A framework for flexible human-machine collaborative systems," in Intelligent Robots and Systems (IROS), 2014 IEEE/RSJ International Conference on, Sep 2014.

[20] I. Millington and J. Funge, Artificial intelligence for games. CRC Press, 2009.

[21] 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.

[22] J. Tang, S. Miller, A. Singh, and P. Abbeel, "A textured object recognition pipeline for color and depth image data," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA), 2012.