A Distributed Control Architecture for Collaborative Multi-Robot Task Allocation

Janelle Blankenburg[1], Santosh Balajee Banisetty[2], S. Pourya Hoseini Alinodehi[3], Luke Fraser[4], David Feil-Seifer[5], Monica Nicolescu[6], and Mircea Nicolescu[7]

***Abstract*— This paper addresses the problem of task allocation for multi-robot systems that perform tasks with complex, hierarchical representations which contain different types of ordering constraints and multiple paths of execution. We propose a distributed multi-robot control architecture that addresses the above challenges and makes the following contributions: i) it allows for on-line, dynamic allocation of robots to various steps of the task, ii) it ensures that the collaborative robot system will obey all of the task constraints and iii) it allows for opportunistic, flexible task execution given different environmental conditions. This architecture uses a distributed messaging system to allow the robots to communicate. Each robot uses its own state and team member states to keep track of the progress on a given task and identify which subtasks to perform next using an activation spreading mechanism. We demonstrate the proposed architecture on a team of two humanoid robots (a PR2 and a Baxter) performing hierarchical tasks.**

I. Introduction

In this paper we propose a control architecture for collaborative multi-robot systems [1], focusing on the problem of task allocation under hierarchical constraints imposed on a joint task. Real-world tasks are not only a series of sequential steps, but typically exhibit a combination of multiple types of constraints, with parts of the task that are sequential, others that have no ordering constraints, and others that may allow for alternative paths of execution. These tasks pose significant challenges even in the single robot domain, as enumerating all the possible ways in which the task can be performed can lead to very large representations and keeping track of the task constraints during execution is not trivial. In previous work [2], [3] we developed an architecture that provides a compact encoding of such tasks, and validated it in a single robot domain.

In this work, we extend the above architecture to address the problem of representing and executing similarly structured tasks in a collaborative multi-robot setting. In this setup the robots can work together, performing individual task steps in order to accomplish the overall task. This poses a new set of challenges pertaining to task allocation, as the robots need to decide on which step of the task to work on and in what order, such that the overall constraints are obeyed, all the required steps are executed, and no robots work on the same part of the task.

To address the above challenges we developed a distributed multi-robot control architecture that makes several key contributions. First, the architecture allows for on-line, dynamic allocation of robots to various steps of the task. This is achieved through a distributed communication mechanism between the robots. Each robot has its own individual copy of the joint task and each node in the task representation communicates directly with its peer nodes on the other robots, sharing information regarding its current state or progress, enabling the robots to know at all times which subtasks are in progress or have been completed by other agents. Second, our architecture ensures that the collaborative robot system will obey all of the task constraints. For this, each robot uses its own state and team member states to keep track of the progress on a given task and uses an activation spreading mechanism to identify which subtasks to perform next, enforcing task constraints. Third, the proposed architecture allows for opportunistic and flexible task execution given different environmental conditions. The activation spreading mechanism enables each robot to select the task steps that are most relevant or easier to perform from its own perspective (e.g. objects that are closer are better than those that are farther away). As our experimental results show, the robots choose to perform different steps given different environmental setups for the same joint task, indicating that the robot team can adapt to varying environmental conditions.

The remainder of the paper is structured as follows: Section II presents related work, Section III presents the technical details of our approach, and Section IV shows the results of our evaluation of the multi-robot architecture. Section V provides details of our ongoing and future work, and Section VI concludes the paper.

II. Related Work

Multi-robot systems gained momentum in the 80's and 90's when a series of projects were implemented successfully such as ACTRESS [4], ALLIANCE [5] and MURDOCH [6]. These projects proposed the efficient use of multi-robot systems over a single powerful robot. To date, a wide range of distributed approaches have been developed for task allocation in multi-robot systems.

Several approaches fall under the category of behavior-based systems [7]. These approaches perform computations

Department of Computer Science & Engineering University of Nevada, Reno, Reno, NV, 89557
[1]Email: jjblankenburg@nevada.unr.edu
[2]Email: santoshbanisetty@nevada.unr.edu
[3]Email: hoseini@nevada.unr.edu
[4]Email: fraser@nevada.unr.edu
[5]Email: dave@cse.unr.edu
[6]Email: monica@cse.unr.edu
[7]Email: mircea@cse.unr.edu

on internal representations in order to decide what action to take. They consist of a collection of parallel, concurrently executing behaviors devoid of a centralized arbiter [8]. Our proposed architecture is such a behavior-based system, relying on activation spreading and peer-behavior communication for task allocation. Parker *et al* [5] proposed one of the first behavior-based architectures for the multi-robot task allocation problem called ALLIANCE and a related L-ALLIANCE architecture [9]. These approaches focus on fault tolerant and efficient control. Werger [10] presents a distributed behavior based approach to the problem of Cooperative Multi-Robot Observation of Multiple Moving Targets (CMOMMT). The architecture uses cross inhibition and cross subsumption between peer behaviors on each robot in order to determine allocation of robots to targets. Unlike these approaches, our architecture incorporates various types of ordering constraints and multiple paths of execution which allows for a more diverse application to multi-robot collaboration tasks, such as building or manufacturing instead of navigation type tasks as in these earlier approaches.

Other approaches focus on a market-based architecture for allocating tasks distributively. In these approaches, the team seeks to optimize an objective function based upon individual robot utilities for performing particular tasks [11]. Gerkey *et al* [6] proposed a novel dynamic task allocation approach for a group of heterogeneous robots utilizing a publish/subscribe messaging system to carry out auctions called MURDOCH. Wang *et al* [12] proposed a market-based task allocation algorithm which utilizes a task evaluation function based on distance fitness and urgency. Trigui *et al* [13] proposed two auction-based distributed algorithms for task allocation namely DMB and IDMB, and found that IDMB (extension of DMB) resulted in nearly optimal solutions and produced an optimal solution in several cases. Unlike these approaches, our approach does not use a complicated utility function or an explicit auction system with a coordinator and bidders. Our hierarchical architecture uses activation-spreading based on distance to the robots' grippers to identify which tasks to complete.

Compared with the above approaches, our architecture focuses on tasks with significant constraints, allowing for both sequential and non ordering constraints, as well as multiple paths of execution to be present in the same task. Our proposed method enables the team of robots to dynamically allocate robots to sub-tasks, while maintaining all the required constraints.

III. Distributed Multi-Robot Architecture

The proposed architecture uses a behavior-based paradigm [7], which allows communication and connectivity between nodes in the architecture. This is built as an extension of our single-robot architecture described in [2] and [3]. The representation enables the system to encode tasks involving various types of constraints such as sequential, non-ordering, and alternative paths of execution. All of these constraints can be incorporated into a single task representation such as that presented in Figure 1. In this example, the *THEN* nodes represent sequential constraints, the *AND* represents non-ordering constraints, and the *OR* represents alternative paths of execution. The setup of this architecture on a single robot is defined in more detail in Section III-A. The extension of this architecture to the multi-robot domain is described in Section III-B.

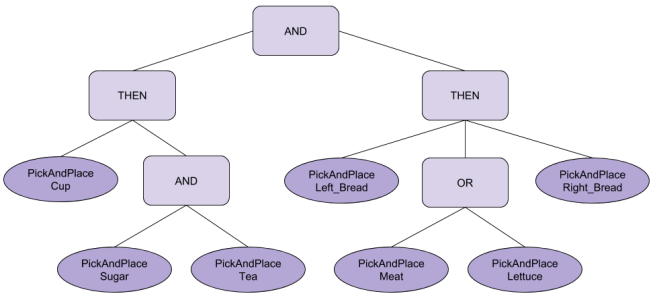

Fig. 1. The full task structure of the tea-time task experiment. Both robots have an identical copy of this task tree. The lighter purple nodes represent the goal nodes of the task structure and the darker purple nodes represent the behavior nodes.

A. Single Robot Architecture

The hierarchical control architecture for single robot scenarios is described in detail in [2]. Below we provide a brief description. To encode a task on a single robot we define two types of nodes:

- **Goal Nodes:** These provide the base goal control behaviors of the hierarchical task structure, and include the *THEN*, *AND*, and *OR* nodes that are used internally in the tree to encode the task constraints:
  - **THEN:** This is a n-ary node which is used to encode sequential constraints (the left child must execute before the children to its right can execute).
  - **AND:** This is a n-ary node which is used to encode non-ordering constraints (children can be executed in any order).
  - **OR:** This is a n-ary node which is used to encode alternative paths of execution (only one of the children will be executed).
- **Behavior Nodes:** These are the leaf nodes in the task tree structure and encode the physical behaviors that the robot can perform, e.g. a *PickAndPlace(Cup)* behavior will control the arm of the robot to pick up a cup from the table in front of it and place it in another location.

In order to maintain communication and connectivity between the nodes in a task tree, each node in our architecture maintains a state consisting of several components:

- **Activation Level:** a number provided by the node's parent and represents the priority placed on executing and finalizing a given node.
- **Activation Potential:** a number representing the node's perceived efficiency, which is sent to the parent of the node.
- **Active:** a boolean variable that is set to true when the node's activation level exceeds a predefined threshold, indicating that the behavior is currently executing.

- **Done:** a boolean variable that is set to true when the node has completed its required work.

The above state information is continuously maintained for each node and is used to perform top-down and bottom-up activation spreading that ensures the proper execution of the task given the constraints. To execute a task, *activation spreading messages* are sent from the root node of a task toward its children. These messages spread the *activation level* throughout the task tree in a top-down manner. At the same time, each node sends *status messages*, which encode a node's current state, to its parent node. These messages spread the *activation potential* throughout the tree in a bottom-up fashion. The state of each node in the task structure is maintained via an update loop which runs at each cycle. This loop performs a series of checks of the node's state and updates the various components of the state accordingly. The full algorithm of the update loop is presented in [2]. We provide the algorithm for the multi-robot domain in Algorithm 1 below.

B. Multi-Robot Architecture

To extend our single robot architecture to the multi-robot domain, several components need to be added, as described below. These are extensions of the ideas proposed in our previous work [3].

1) Task Representation for Multiple Robot Domain: In a multiple robot scenario, each robot has its own instance of the task tree structure, identical to that of the other robots, which encodes the joint team task. Equivalent nodes in the task structures across robots are called *peers*. These peers are the means of communication between the robots and allow nodes to keep track of other robots' progress on the task. While the task hierarchy is uniform across robots, the *activation potential* and *activation levels* for each node are calculated individually by each robot.

In addition to the state components used in the single robot case above, the multi-robot state of each node contains two new variables:

- **peer_active**: a boolean variable that is true when either the node is active or the node's peer is active.
- **peer_done**: a boolean variable that is true when either the node is done or the node's peer is done.

These additional state variables are required for collaboration between the robots because they allow each robot to identify if the node is currently being worked on or was already completed by another robot. This information is necessary to ensure there is no overlap in the sub tasks that the robots perform. By identifying what tasks are being worked on by its teammates and which tasks are already completed, each robot is able to determine the next step it should perform based on the activation spreading mechanism within its own task tree structure as well as its own state.

2) Peer Message Passing for Robot Communication: In order to communicate across robots we use a distributed message passing system called ZeroMQ [14], which opens a channel allowing messages to be passed between each set of peer nodes in a given task tree structure. Each node continuously passes *status messages* to its peer nodes through the ZeroMQ interface. These *status messages* contain the same encoding of a node's state as the messages used in the bottom-up activation spreading in the single robot case. The same encoding is used here because the peer portions of the state variables are not needed as the peer nodes receiving these messages already know their own states. By continuously passing this information between nodes, all nodes in the tree are able to keep track of their peers' progress and can use this data to update their own states accordingly.

3) Decision Making Process for Task Allocation: In order to decide which part of the task to work on, each robot runs a state update loop that is performed on every node in its task tree.

Algorithm 1 Update Loop

```
1:  if done == FALSE then
2:    if active == TRUE then
3:      if Precondition() == TRUE then
4:        Activate()
5:      else
6:        SpreadActivation()
7:      end if
8:      ActivationFalloff()  // decays by α ∗ activation_level
9:    end if
10: end if
```

The details of this update loop are shown in Algorithm 1. This process is responsible for ensuring that the nodes are activated in a way that obeys all of the task constraints encoded by the architecture and that the robots do not choose to work on the same part of the task. In order to start the task execution, a positive *activation level* is passed externally to the root of each robots' task tree. Once this happens, the activation spreading mechanism described in the single robot case propagates this activation to the rest of the nodes. As the activation spreads from the root of the task tree, the *activation levels* of the top node in each task tree will rise above a given threshold, which causes it to be set to *active*. This allows the node's state to pass the *done* and *active* checks (lines 1, 2) of this loop and follow the remaining logic of the algorithm to determine which node to activate next in each tree. As the loop runs, if any nodes are already *active* or *done*, the update loop will not activate them again. This ensures that there is no repeating activation sent to already executed nodes, so that they will only be performed once. Once a node is *active*, it checks the preconditions of the node. Preconditions are the set of conditions that must be completed prior to a node beginning its work. These conditions ensure that work is only started after all the required task constraints on a node are satisfied. If the preconditions are met we run the *Activate* function. If the preconditions are not met we spread activation across the other nodes in the calling node's own task tree in the same manner as the single robot case. At the end of the loop, the activation falloff function will lower the *activation level* of

the current node to ensure that nodes that are not currently being considered for work are less likely to be performed next than nodes that are currently being given attention by the control architecture.

Algorithm 2 Activate

```
1: if peer_check_thread == FALSE then
2:     check_peer = TRUE
3:     peer_check_thread = new boost thread
4:     peer_check_thread → detach
5: else if check_peer == FALSE then
6:     peer_check_thread → interrupt
7:     peer_check_thread = NULL
8: end if
9: if peer_okay == TRUE then
10:    if active == FALSE
       AND done == FALSE then
11:        if ActivationPrecondition() == TRUE then
12:            lock(work_mutex)
13:            active = TRUE
14:            PublishStateToPeers()
15:        end if
16:    end if
17:    peer_okay = FALSE
18: end if
```

The *Activate* function is described in Algorithm 2. This function uses the *Peer Check Thread* to check the status of the calling node's peers. This thread runs asynchronously from the *Activate* function. Upon completion, the thread sets a condition variable *check_peer* to false so that the next call to the *Activate* function will know to stop the current thread as shown in lines 5-7 of the algorithm. If the thread no longer exists, the *Activate* function launches a new thread in order to restart the check of the peers' states (lines 1-4). During the peer checking, the check peer thread updates the *peer_okay* state variable accordingly. If it was determined that the peers of the calling node are in an acceptable state (i.e. not *active* and not *done*), the *Activate* function then begins checking the state components of the calling node's own state, i.e. *active* and *done*. If the node is not already *active* nor *done*, and the preconditions of the node are met, the *Activate* function locks the work mutex, sets the node to *active*, and publishes this updated state to its peers via the ZeroMQ channels described above. The work mutex ensures that only one node of the task tree is working at a time. Lastly, this function sets *peer_okay* to false so the peer check thread can update this as it sees fit on the next call to this function.

The *Peer Check Thread* function is described in Algorithm 3. This function checks the peer components of the calling node's state to ensure the work being done by the peers does not overlap with the work being done by the calling node. The thread uses the lock on the peer mutex to force this thread to wait until the *Activate* function is ready to check the status of the node's peers. At this time, *check_peer* gets set to true and the thread begins the checking procedures. The thread first publishes the node's state to its peers via

Algorithm 3 Peer Check Thread

```
1: lock(node → peer_mutex)
2: // wait to check peer's status until asked
3: while check_peer == FALSE do
4:     node → cv.wait(lock)
5: end while
6: PublishStateToPeers()
7: sleep one loop
8: for each peer do
9:     if peer_done == TRUE then
10:        peer_okay = FALSE
11:    else if peer_active == TRUE then
12:        lower activation level
13:        peer_okay = FALSE
14:    else if peer_active == FALSE
         AND peer_done == FALSE then
15:        peer_okay = TRUE
16:    end if
17: end for
18: check_peer = FALSE
```

ZeroMQ. This ensures that the peers have the most recent state of the current node so they can update themselves accordingly during their own checks. The thread then sleeps for one cycle of the update loop. During this time, the peers send their own current states to the calling node via ZeroMQ. After one loop, the calling node will have received updated states from all of its peers, which it uses to set the *peer_okay* variable. The thread iterates through the list of the node's peers and sets *peer_okay* accordingly. If any of the peers are *done* or already *active*, *peer_okay* is set to false. This means that one of the peers has already completed this part of the task or is currently working on it, so the calling node cannot work on this part. In addition if *peer_active* is true, we want to deter the robot from working on other nodes in this portion of the task tree by lowering the node's *activation level*. Since some other robot is already working in this area in this case, working in a different area is less likely to result in work being paused due to sequential constraints. If the peer is not *active* nor *done*, then the calling node is able to begin work on this node and thus *check_peer* is set to true. Lastly, the thread sets *check_peer* to false. In this way the next call to the *Activate* function will know that the thread has finished its checks, has updated *peer_okay* accordingly, and is ready to be stopped and relaunched.

Together, the above three algorithms maintain and communicate the states of all of the nodes to their corresponding peer nodes on the other robots in order to ensure that the robots can work collaboratively to complete the task in a manner that follows its constraints.

IV. Experimental Evaluation

In order to verify and demonstrate our multi-robot hierarchical control architecture we implemented it on a PR2 robot and a Baxter robot to jointly perform a task that exhibits all the constraints (sequential - THEN, non-ordering - AND,

and alternative paths - OR), using *PickAndPlace* behaviors as a basic behavior. The implementation of the *PickAndPlace* behaviors and their integration with our architecture are described below in Section IV-A. The experiments used to evaluate the architecture are discussed in Section IV-B.

A. Robot Implementation

1) Pick and Place: In order to validate our architecture we implemented a pick and place system which converts the activation of the *PickAndPlace* behavior nodes in our task structure to physical actions on the robots, allowing them to grasp and place various objects at desired locations. To get the robots to move to the correct position of the objects in a given task, we pass commands to the robots via MoveIt [15] through its interface with the Robot Operating System (ROS) [16]. The *PickAndPlace* behavior is implemented in such a way that when the task structure activates a *PickAndPlace* behavior node, it will move to the corresponding object, pick it up, and place it at another location. For the purpose of the pick and place movement, the right arm on each robot was used. The full picking and placing movement is the required work that must be completed in order for the *PickAndPlace* node to be marked as *done*. The architecture waits until the place command has finished before it activates another node, since only one node per robot can be doing work at any given time.

In our implementation, between the pick and place parts of the *PickAndPlace* behavior the gripper returns to a location that has a small offset above and to the right of the pick/place location of the object the robot is trying to grasp. This allows a human watching the task to infer which object the robot is reaching for. In addition, this allows us to incorporate the distance between the robot's gripper and each object into the *activation potential* to make the robot more likely to pick up objects closer to its current position, thereby making the overall pick and place task more efficient. In order to identify the objects to be grasped, in our current set up we use predefined locations and orientations for the objects for the Baxter. On the PR2 the orientations are predefined but the locations for the PR2 are received through ROS service calls through the robot's vision system.

2) Vision System: The vision system captures video streams from the Kinect V1 camera on the PR2's head. It performs object detection by using a Mixture of Gaussian [17] background subtraction technique on the depth plane of the data. During this processing, the scene is assumed to be static. Morphological opening followed by morphological closing [18] clean the foreground map, removing small noisy areas. Separate foreground regions are assigned different labels by computing connected components. Features of the intensity plane obtained from the camera's RGB stream are used to detect objects. For each object in a given task, a Histogram of Oriented Gradients (HOG) [19] is used to describe the object's shape, while a normalized color histogram of the object describes its color content. A Support Vector Machine (SVM) [20] is then used to classify the detected objects based on their extracted features. Figure 2 shows an example of a set of objects being detected and recognized using the vision system. Here, the 3D point cloud from the Kinect is used to compute the 3D location of each object's centroid, which is then passed as the picking location of the object in the pick and place system via the response of the ROS service call from the pick and place portion. The integration of the vision system with the task architecture on the Baxter robot is currently in progress.

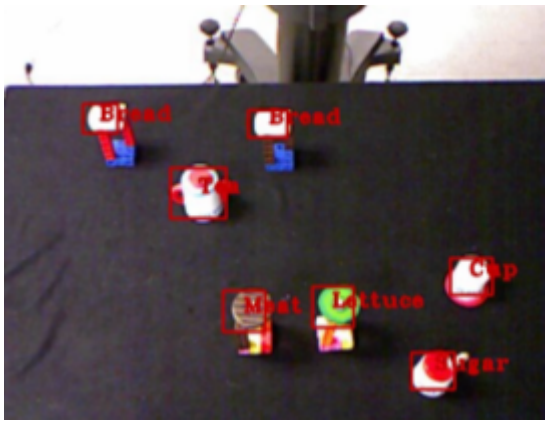

Fig. 2. An example detection and recognition by the vision part using the Kinect v1 camera on the PR2 taken during scenario 3 of the tea-time experiments.

B. Experiments

We demonstrate our multi-robot control architecture with two sets of tasks designed to illustrate the key contributions of the architecture: dynamic task allocation, obeying of task constraints by the robot team, and opportunistic, flexible execution in different environmental conditions. The first set of tasks are meant to validate the correctness of each of the goal nodes individually. The second set illustrates that these goal nodes can be combined together to encode a complicated joint task. These experiments were run on a PR2 and a Baxter facing each other with a table and a set of objects in between them as shown in Figures 3 and 4.

1) Simple Behavior Experiments: The first set of experiments consists of three different tests, one for each base goal node. The task encodings for these tests were:

- (*THEN PickAndPlace(Left_Bread) PickAndPlace(Meat) PickAndPlace(Lettuce) PickAndPlace(Right_Bread)*)
- (*AND PickAndPlace(Left_Bread) PickAndPlace(Meat) PickAndPlace(Lettuce) PickAndPlace(Right_Bread)*)
- (*OR PickAndPlace(Left_Bread) PickAndPlace(Meat) PickAndPlace(Lettuce) PickAndPlace(Right_Bread)*).

For each test case we set the four different objects from a children's toy set (Left_Bread, Meat, Lettuce, and Right_Bread) on the table in between the robots to be picked and placed to a goal location. We chose these four objects because they can be easily grasped by both robots. The two bread slices were encoded to be *Left* or *Right* with respect to their position from the PR2's viewpoint. We tested each goal node task in three different setups, varying the locations

Fig. 3. A view of each of the three setups for the simple behavior experiments. Each scenario has different locations for the objects as well as different initial positions for the right gripper of each robot.

TABLE I

THE RESULTS OF THE SIMPLE NODE EXPERIMENTS. THE COLUMNS ARE THE SCENARIO NUMBERS AND THE ROWS ARE THE GOAL NODE TYPE.

	Scenario 1	Scenario 2	Scenario 3
THEN	Baxter:Left_Bread, Baxter:Meat, PR2: Lettuce, PR2:Right_Bread,	PR2:Left_Bread, PR2:Meat, Baxter:Lettuce, Baxter:Right_Bread,	Baxter:Left_Bread, PR2:Meat, Baxter:Lettuce, PR2:Right_Bread,
AND	Baxter:Left_Bread, PR2:Lettuce, Baxter:Meat, PR2:Right_Bread	Baxter:Lettuce, PR2:Left_Bread, Baxter:Right_Bread, PR2:Meat	Baxter:Left_Bread, PR2:Meat, Baxter:Lettuce, PR2:Right_Bread
OR	Baxter:Left_Bread	Baxter:Lettuce	PR2:Meat

for each one. The view of each setup is shown in Figure 3. In scenario 1, Left_Bread and Meat were close to the Baxter and Lettuce and Right_Bread were close to the PR2. Scenario 2 swapped the locations so that Left_Bread and Meat were closer to the PR2 and Lettuce and Right_Bread were closer to the Baxter. Lastly, scenario 3 had all four objects lined up in a row along the center of the table. For scenarios 1 and 2, the place locations were in between and slightly in front of the two objects closest to each robot. For scenario 3, the place locations were between and slightly behind the two objects on each robot's right side. These place locations are important due to the fact that the activation spreading mechanism uses the distance from the right gripper of the robot to determine which object to pick next. These different scenarios illustrate that the paths through the task tree generated by the architecture in each case follow the constraints of the respective goal nodes in different environmental conditions.

2) Simple Behavior Results: The resulting order of execution of the behavior nodes in each scenario for each goal node task are displayed in Table I. The columns are the scenario numbers and the rows are the goal node type. The cells indicate the order in which the four objects were picked up along with which robot picked them up. Each line in a cell represents the object(s) picked up during one iteration of pick and place on either robot.

For the THEN trials, only one object was picked up at a time by either robot which required four iterations of pick and place to finish. For each of these scenarios we see that the objects get picked in the same sequential order, but by different robots depending on their distances from the robots' right grippers. In these trials, only one robot is picking an object at a given time due to the sequential constraints of the *THEN*. In scenario 1, the Baxter picks the first two objects and then the PR2 picks the second two. These tasks are swapped in scenario 2. In scenario 3, we see that the Baxter and PR2 take turns picking the objects. This shows that the sequential constraints of the *THEN* node hold irrespective of the object placements.

For the AND trials, both robots were able to grasp objects simultaneously and thus these only required two iterations of pick and place to finish. For these tasks we see that the objects are not necessarily picked up in sequential order, since the AND does not encode ordering constraints. Instead, the robots simultaneously pick up the objects closest to their right grippers first, and then move on to the next closest objects. Since there are no ordering constraints, the robots are able to pick objects at the same time.

The OR trials only selected one object and so they required one iteration. The scenarios for the *OR* node only pick the object closest to either of the robots' right grippers, since only one child needs to be performed for the *OR* node's constraints to be satisfied. Since different locations correspond to different objects being picked, we see that the multiple paths of execution through things along an *OR* node are constrained correctly.

Since each scenario placed the objects in different locations and each node type encodes different ordering constraints, we see that the tasks were completed in different orders and with a different number of iterations of pick and place. The above test cases show that the robots are able to collaboratively complete a given task, allocating sub-tasks according to the current environmental conditions.

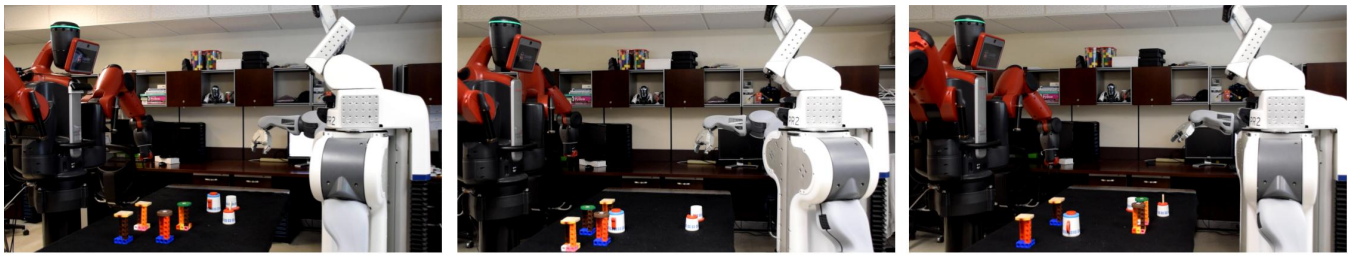

Fig. 4. Views of the initial setups of the three tea-time task experiments. The object locations are different in each scenario.

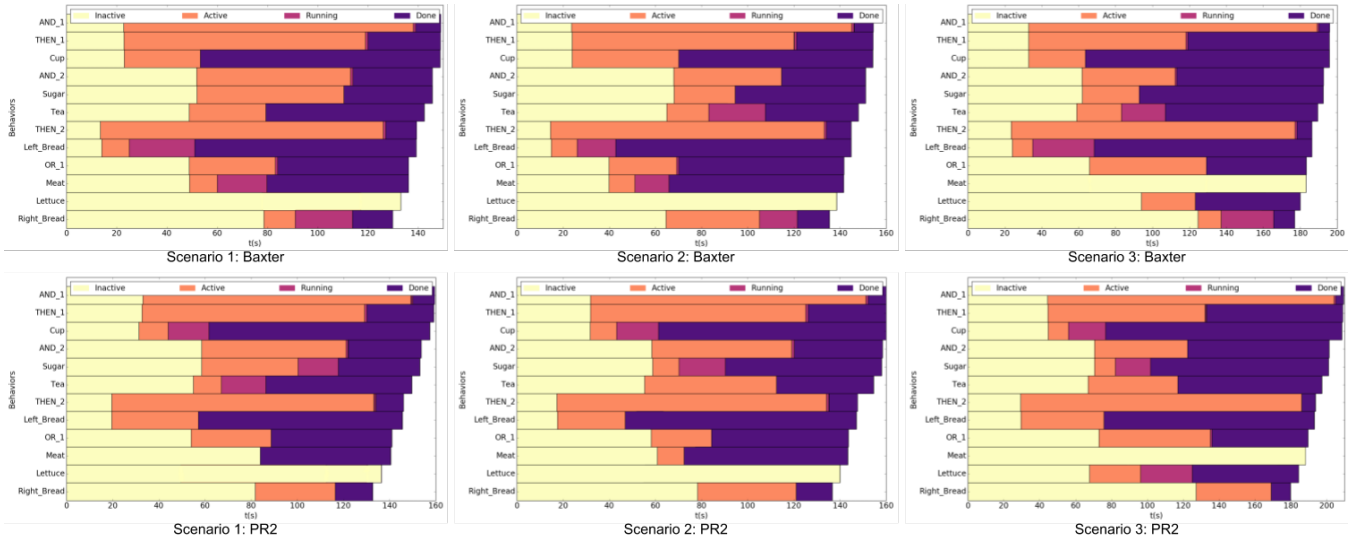

Fig. 5. The timing diagrams of the tea-time task scenarios on the PR2 and the Baxter. Each column corresponds to a different scenario. The top row are the diagrams for the PR2 and the bottom row are for the Baxter. These diagrams represent the times at which the state of a node in a given task tree changed. Within each graph, each row corresponds to a behavior node named as its corresponding object. The horizontal axis is increasing time.

3) Complex Tea-Time Experiments: The second set of experiments consisted of an encoding of a complex task structure, which we call tea-time. The task structure for this experiment is shown in Figure 1. This structure consists of two main tasks: making a sandwich and making tea. Each task is made up of several sub-tasks. The tea task corresponds to the left branch off the topmost *AND* node in the task tree. The sandwich task corresponds to the right branch of the topmost *AND* node. We ran three scenarios of this task structure with the objects in different locations. This illustrates that the control architecture can determine different paths through the same task tree based on the locations of the objects and their corresponding constraints in the architecture. The setups for the three experiments are shown in Figure 4. For each scenario, the placing position of the objects for the tea task are to the PR2's right (the Baxter's left) and the placing locations of the objects for the sandwich are to the PR2's left (the Baxter's right).

4) Complex Tea-Time Results: The complex tea-time experiment was demonstrated on three different scenarios, each with different initial object locations. The timing diagrams illustrating the change of state of each node in the task structure for both robots are shown in Figure 5. The different color bars in the figure represent the times during which a particular behavior node is in one of the following states: inactive, active, running or done. The intervals corresponding to the running state identify when a given *PickAndPlace* behavior is being executed and are thus indicative of the order in which various sub tasks have been performed. We see from scenario 1 that the robots worked on the nodes corresponding to the task with the objects closest to them; the PR2 worked on the tea task and the Baxter worked on the sandwich. This illustrates that multiple robots are able to collaborate on an overall task by completing the different main sub tasks of that task. In scenario 2 we see that the Baxter completes the sandwich task and one part of the tea task. In scenario 3 we see that the Baxter and the PR2 both perform one portion of the opposite robot's main sub task (i.e. the Baxter grabs the Tea and the PR2 grabs the Lettuce). These two scenarios demonstrate that the architecture allows multiple robots to share the sub tasks to varying degrees in order to complete the overall task, thereby highlighting the extent to which collaboration is possible in the proposed multi-robot control architecture. These diagrams show that the architecture adheres to all the constraints while performing the joint task allocation

between the robots regardless of object locations.

V. Future Work

In order to further demonstrate the capabilities of this architecture there are several implementation aspects that we are currently pursuing. First, we are currently working on integrating the vision system with the architecture on the Baxter robot. We are also exploring incorporating collision avoidance on the Baxter and PR2 in our motion planning for the *PickAndPlace* behaviors. This will allow the robots to choose objects much closer to each other. In turn, having the capability for the robots to maneuver in close proximity to each other opens the door for more explicitly collaborative tasks such as hand-offs between robots. One idea for the hand-off behavior that we are currently investigating is a bucket-brigade task. This task will require stricter timing constraints of sub-tasks, collaboration between the robots to complete the task, and have actions which can be completed by one robot, but not another.

VI. Conclusion

This work proposes a distributed control architecture for multi-robot systems that perform tasks with complex, hierarchical representations, which contain different types of ordering constraints and multiple paths of execution. The architecture provides several key contributions. First, it allows the robots to dynamically decide on which part of the joint task to work on. For this, the architecture uses a distributed message passing system for communication between the robots. Each robot has a task tree encoding of the given task and communicates with its teammates as well as within other parts of its own task tree to identify which parts of the task are currently being performed as well as those that have already been completed. Second, the control architecture allocates tasks in such a way that the robots, working together to complete the overall task, adhere to all task constraints. These constraints can be sequential, non-ordering, or require alternative paths of execution. Third, as shown by our experimental evaluation, the architecture allows for opportunistic task execution on joint tasks given different environmental conditions. We demonstrated the performance of our collaborative control architecture on two sets of experiments using a PR2 robot and a Baxter robot, performing tasks that contain individual as well as combinations of all of the above mentioned constraints.

Acknowledgment

The authors would like to acknowledge the financial support of this work by Office of Naval Research (ONR, #N00014-16-1-2312, #N00014-14-1-0776).

References

[1] T. Arai, E. Pagello, and L. E. Parker, "Advances in multi-robot systems," *IEEE Transactions on robotics and automation*, vol. 18, no. 5, pp. 655–661, 2002.

[2] L. Fraser, B. Rekabdar, M. Nicolescu, M. Nicolescu, D. Feil-Seifer, and G. Bebis, "A compact task representation for hierarchical robot control," in *International Conference on Humanoid Robots*, (Cancun, Mexico), pp. 697–704, IEEE, November 2016.

[3] L. Fraser, B. Rekabdar, M. Nicolescu, M. Nicolescu, and D. Feil-Seifer, "A hierarchical control architecture for robust and adaptive collaborative robot task execution," in *Robotics: Science & Systems: Workshop on Planning for Human-Robot Interaction: Shared Autonomy and Collaborative Robotics*, (Cambridge, MA), June 2016.

[4] H. Asama, A. Matsumoto, and Y. Ishida, "Design of an autonomous and distributed robot system: Actress.," in *IROS*, vol. 89, pp. 283–290, 1989.

[5] L. E. Parker, "Alliance: An architecture for fault tolerant, cooperative control of heterogeneous mobile robots," in *Intelligent Robots and Systems' 94.'Advanced Robotic Systems and the Real World', IROS'94. Proceedings of the IEEE/RSJ/GI International Conference on*, vol. 2, pp. 776–783, IEEE, 1994.

[6] B. P. Gerkey and M. J. Matarić, "Murdoch: Publish/subscribe task allocation for heterogeneous agents," in *Proceedings of the International Conference on Autonomous Agents*, pp. 203–204, ACM, 2000.

[7] R. C. Arkin, *An Behavior-based Robotics*. Cambridge, MA, USA: MIT Press, 1st ed., 1998.

[8] M. J. Mataric, "Behavior-based control: Main properties and implications," in *Proceedings, IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems*, pp. 46–54, 1992.

[9] L. E. Parker, "L-alliance: Task-oriented multi-robot learning in behavior-based systems," *Advanced Robotics*, vol. 11, no. 4, pp. 305–322, 1996.

[10] B. B. Werger and M. J. Mataric, "Broadcast of local eligibility: Behavior-based control for strongly cooperative robot teams," in *Proceedings of the Fourth International Conference on Autonomous Agents*, AGENTS '00, (New York, NY, USA), pp. 21–22, ACM, 2000.

[11] L. E. Parker, "Multiple mobile robot systems," in *Springer Handbook of Robotics* (B. Siciliano and O. Khatib, eds.), pp. 921–941, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[12] Z. Wang, M. Li, J. Li, J. Cao, and H. Wang, "A task allocation algorithm based on market mechanism for multiple robot systems," in *Real-time Computing and Robotics (RCAR), IEEE International Conference on*, pp. 150–155, IEEE, 2016.

[13] S. Trigui, A. Koubaa, O. Cheikhrouhou, H. Youssef, H. Bennaceur, M.-F. Sriti, and Y. Javed, "A distributed market-based algorithm for the multi-robot assignment problem," *Procedia Computer Science*, vol. 32, pp. 1108–1114, 2014.

[14] H. Pieter, "Zeromq: messaging for many applications," *O'Reilly Media*, p. 484, 2013.

[15] I. A. Sucan and S. Chitta, "Moveit!," *Online at http://moveit. ros. org*, 2013.

[16] 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, 2009.

[17] C. Stauffer and W. Grimson, "Adaptive background mixture models for real-time tracking," in *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, IEEE Comput. Soc, 1999.

[18] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (4th Edition)*. Pearson, 2017.

[19] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2005.

[20] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, sep 1995.