Single Robot Multitasking Through Dynamic Resource Allocation

Tyler Becker[1], Song Jiang[1], David Feil-Seifer[2], Monica Nicolescu[1]

***Abstract*— This paper addresses the problem of dynamic allocation of robot resources to tasks with hierarchical representations and multiple types of execution constraints, with the goal of enabling single-robot multitasking capabilities. Although the vast majority of robot platforms are equipped with more than one sensor (cameras, lasers, sonars) and several actuators (wheels/legs, two arms), which would in principle allow the robot to concurrently work on multiple tasks, existing methods are limited to allocating robots in their entirety to only one task at a time. This approach employs only a subset of a robot's sensors and actuators, leaving other robot resources unused. Our aim is to enable a robot to make full use of its capabilities by *having an individual robot multitask*, distributing its sensors and actuators to multiple concurrent activities. We propose a new architectural framework based on Hierarchical Task Trees that supports multitasking through a new representation of robot behaviors that explicitly encodes the robot resources (sensors and actuators) and the environmental conditions needed for execution. This architecture was validated on a two-arm, mobile, PR2 humanoid robot, performing tasks with multiple types of execution constraints.**

2023 IEEE-RAS 22nd International Conference on Humanoid Robots (Humanoids) | 979-8-3503-0327-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/HUMANOIDS57100.2023.10375225

I. INTRODUCTION

In this paper we focus on the problem of adaptive task allocation, from the perspective of complex robotic platforms with multitasking capabilities (such as mobile humanoid robots), working on tasks with hierarchical representations and multiple types of execution constraints.

Previous work on architectural decision making and control focuses primarily on encoding tasks using representations that are compact, modular, flexible and embed the necessary temporal constraints for a given task. However, these existing architectures do not account for situations (sometimes only detected at run-time) in which the efficiency of the task execution could be improved by performing multiple subtasks concurrently. These methods consider each basic control module as having full control of the entire robot (sensors, actuators) so that collisions between subtasks cannot happen and therefore any multitasking, if considered, is implemented explicitly, at the time when the controller is designed. For instance, a two-arm humanoid robot (Fig. 1) would be underutilized if it were only working on a task that requires only one arm because the other arm could have been used in parallel for a different purpose (e.g., to pick up two objects concurrently). Existing task representations to date

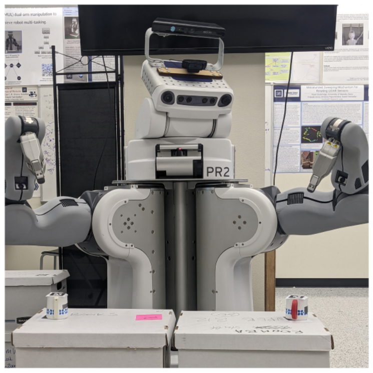
Fig. 1. This figure shows a PR2 robot picking up both of the cups in front of it. The PR2 has two arms and they are both free, so it should pick up both cups simultaneously.

do not provide the mechanisms needed to support adaptive, run-time robot multitasking.

In contrast with existing approaches that consider a robot to be an entity that is entirely allocated to a single control module (subtask) at any given time, we view a robot to be a collection of *resources*, such as *actuators* (arms, legs, etc.) and *sensors* (cameras, lasers, etc.). We then consider each subtask as requiring some subset of these resources. This representation transforms the problem of allocating robots to tasks into a problem of allocating resources to tasks. The major challenge that stems from allowing robots to distribute their resources across multiple concurrent tasks is that the physical constraints (number/placement of actuators/sensors) along with the task parameters (instantiated at task execution) both impact the feasibility of allocations, which have to be determined at run-time for a specific task and environment. While highly advantageous, allowing different resources that belong to the same robot to be employed for different tasks requires fundamentally different approaches to allocate robot resources in order to ensure that the allocations are feasible.

The main contribution of this research is a novel autonomous control architecture based on *Hierarchical Task Trees* (HTTs) which employs a new representation of robot behaviors through a specific encoding that explicitly considers the *resources* and environmental conditions needed for execution. Relying on this new representation, our work brings *novel algorithms that enable a single robot to multitask*, allowing it to allocate its resources to multiple, compatible, concurrent tasks. This system was validated with a humanoid PR2 robot in a series of experiments designed to showcase the ability of the robot to dynamically decide when and how to multitask, while ensuring that no conflicting actions are being performed.

The remainder of the paper is organized as follows:

[1]Robotics Research lab, Department of Computer Science & Engineering, University of Nevada, Reno, Reno, NV 89557, USA tbecker@nevada.unr.edu, songjiang@nevada.unr.edu, monica@unr.edu

[2]Socially Assistive Robotics Group, Department of Computer Science & Engineering, University of Nevada, Reno, Reno, NV 89557, USA dave@cse.unr.edu

Section II covers the background and related work, **Section III** describes the functionality of the architecture, **Section IV** covers the design of the experiment, **Section V** reports on our results, **Section VI** provides a brief discussion of the results and **Section VII** concludes the paper.

II. RELATED WORKS

To date, the term multitasking has been mostly employed to refer to single robotic systems with multiple computational modules running concurrently. The most representative examples include the subsumption architecture [1] and behavior based control [2]. In these architectures, multiple behaviors concurrently process inputs from a robot's sensors to provide output for the actuators. However, the tasks are structured such that *only one behavior is pursued by the robot at any given time*. Although all the behaviors are running, all with the exception of one are either not relevant due to preconditions or cannot control the actuators due to suppression of motor commands. Furthermore, the tasks fully specify which actuators/sensors are used for each behavior, without any ability to allocate a robot's resources independently.

There are other task representations frequently used in robotics. A fairly popular representation in the robotics domain and in the video game industry is a Behavior Tree (BT). However, while there has been extensive work on using BTs for autonomous control, there has been little work in the area of multitasking. The research that has been done is primarily focused on creating safe conditions for behaviors in the tree to run in parallel only under predefined parallel nodes. Colledanchise et al. propose two new nodes that provide predictable concurrent behaviors at run time [3]. Concurrent Behavior Trees [4], similar to our work, enable the BT to keep track of resources and of the work completed by tasks that are run in parallel in order to ensure that no behaviors are preempted. However, these architectures still rely on the user to define which behaviors can be run together whereas our focus is on allowing the architecture to decide on any behaviors which could be run in parallel.

Examples also exist of complex articulated robots (such as the mobile humanoid PR2) using both arms simultaneously to fold laundry [5][6]. In these situations, however, the task steps and the actuator allocations are fully specified in the task representation: the arms are used simultaneously and toward the same goal. This project will focus on the ability of a robot to decide when to pursue multiple different goals and when to execute different behaviors simultaneously. There are also examples of using multiple arms to complete tasks quicker both on the same robot [7], or as distributed teams of robots [8]. In both of these cases the ordering of the task, as well as the allocation of the resources or robots is planned prior to the robots starting the task.

Previous work with HTTs [9] has shown that they are an effective way of representing and executing tasks with a variety of temporal constraints under varying environmental conditions. HTTs have been extended to enable both robot-robot [10] and human-robot [11] collaboration, using a theory of mind approach in which collaborators are represented with a copy of the same controller (task tree). In the robot-robot collaborative domain, coordination between teammates is achieved by communication between peer nodes in the corresponding task trees, while in the human-robot domain, the robot maintains a simulated version of a human's mental model of the task. HTTs have also been extended to be able to recover from interrupts or problems which would undo tasks that were considered completed [12]. Because our research is additive in nature, further as well as previous research with HTTs could utilize multi-tasking.

As shown in [13] the MT-SR problems (multitasking robots working on single-robot tasks) are an instance of the Set Partitioning Problem (SPP), which is strongly $\mathcal{NP}$-hard [14]. Heuristic solutions to these problems have been investigated for SPP ([15], [16]), but they perform poorly in the general domain and it is unclear how they can be transitioned to the robot domain.

The focus of this research is to design an architecture that allows a robot to dynamically choose, at run-time, how its own resources should be allocated, based on the current conditions of the environment, in order to execute the task given the specified constraints and in order to enable the robot to multi-task.

III. METHODS

The following section provides an explanation of the proposed architecture. First, we briefly explain the basic functions of a HTT. Then, we describe the changes we made to the representation of the low level behaviors. Finally, we go over the additional logic governing the task nodes.

A. Hierarchical Task Trees

We base this work on the HTT representation introduced in [9]. This architecture provide an abstract representation of a task by splitting it into a series of low-level *behavior nodes*, grouped together such that their ordering constraints are maintained by high-level *task nodes*. Low-level behaviors in general, represent basic time-extended control modules that achieve or maintain certain goals. For example, picking up an object is an example of a low level behavior: all of the steps for the behavior (finding and then grabbing the object) must be performed in order for the goal to be achieved, constituting an atomic module that cannot be divided. In the structure of the tree, these behavior nodes are represented as leaves and they are strung together as children of internal task tree nodes to create more complex, higher level behaviors.

A *task node* provides an implementation of the ordering constraints of the tasks. There are three types of task nodes:

- **AND**: All of the children of an **AND** node must be completed, however, they can be completed in any order.
- **OR**: Only one child of an **OR** node must be completed.
- **THEN**: All of the children of a **THEN** node must be completed and in an order specified prior to starting the task.

HTTs allow for the representation of arbitrarily complex tasks and they enable opportunistic task execution based on the particulars of the robot's environment. This is achieved

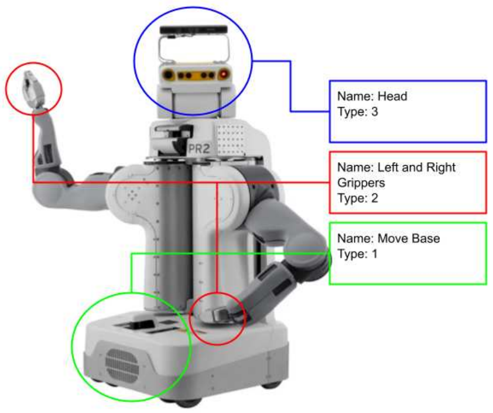

Fig. 2. A visual representation of the resources on the PR2 that were used in this paper. We considered both grippers, the head, and the movable base as resources, however the definition could easily extend to sensors and individual cameras as well.

via a two-way activation spreading mechanism: first, top-down messages are passed from the task nodes to their children in order to enforce the specific execution constraints, and second, bottom-up messages are sent from the low-level behaviors with *activation potential*, representing the perceived efficiency of running a low level behavior at the current time. Each type of low level behavior has its own *activation potential*. Task nodes derive their *activation potential* entirely based on the values of their children. At run-time, the behavior with the highest *activation potential* is the one that is allowed to run, gaining full control of the robot's resources. A complete description of this HTT architecture can be found in [9].

B. Resources and Low Level Behaviors

We implemented three novel mechanisms in the low level behaviors in order to facilitate multitasking: 1) a new way to represent resources on the robot; 2) a way for the behaviors to define what resources they need; and 3) a way for the behaviors to pass access to resources after they finish execution. We imposed a strict definition of a low level behavior to be an action which, if interrupted, must be restarted from its beginning (i.e. picking up an object, or moving to a destination). Altogether, these allow the architecture to pair behaviors that have compatible resource needs while not losing control of which behavior controls which resource.

A resource of the robot has the following properties:

- **Type**: This is to define different groups of actuators and other resources on the robot. An example of the types used in this paper is shown in Fig. 2.
- **Number of Owners**: This is the number of behaviors which are currently using the resource.
- **Channel**: This is the intention with which the resource is being used currently. Behaviors can either require *exclusive* access to a resource, or *share* it with other behaviors that have compatible intentions.
- **Name**: The physical name of the resource.
- **Locked**: This is a boolean value which defines whether or not the resource is in use.

The total canonical state of the resources available on the robot is maintained by the root node of the task tree and is communicated to all the children nodes. As later described in Section III-C, the architecture primarily uses this information to determine which behaviors are compatible to run concurrently.

By using this definition of resources, behaviors can accurately describe how they intend to run, or rather what they intend to use while running. For instance, a behavior to move the robot to a destination requires itself to be the only user of the move base. Resources like sensors and localization information can be shared trivially, however, the navigation can only have a single destination. A behavior to pick up an object requires that the base is completely still during its actions. This means that behaviors to pick up objects are compatible to share control of the base with other nodes that also require the move base locked in place, as long as they require the base to be in the same location.

Furthermore, if behaviors specify what resources they require and their operation mode (shared or exclusive) then the architecture can accurately keep track of which behaviors have control of which resources. For instance, a *pick* behavior must retain control of the gripper which it was allocated because that gripper now has an object in it. This prevents the robot from trying to pick up two objects with the same arm. With the *move* behavior, the situation becomes more complex. First, the *move* behavior should pass its control of the move base on to the behavior which runs after it. This is based on the assumption that the *move* behavior was activated with some subsequent purpose in mind, such as if the next thing the robot needs to do is pick up an object at the destination. This prevents the robot from becoming distracted by some different goal. If the *move* behavior has control of any of the grippers, for example, it can be presumed that the robot was holding something with the intention of performing an action with it at the destination. In that case, the *move* behavior should also hold on to control of those resources. In fact, the *move* behavior should pass on all resources to the next behavior that is activated and has the ordering requirement of being performed immediately after itself. This is again based on the assumption that there is some purpose beyond just moving to the destination.

For the *move* behavior, we calculate the activation potential as the inverse of the distance from the robot to the goal which is then scaled based on the availability of the grippers. This is to make sure that closer destinations are preferred only if the *move* behavior in question has access to resources which it needs at the destination. A formal definition for the *move* behavior's activation potential is as follows:

$$ap = \frac{n}{||x_o - x_d||} \tag{1}$$

Where n is the number of grippers available to the behavior (e.g. the PR2 has two grippers available at the start of a task), x_o is the current position of the robot and x_d

is the destination of the *move* behavior. In order to run, the *move* behavior requires exclusive control of the move base.

For the *pick* and *place* behaviors we assign a constant high value so that they would always supersede the *move* behaviors if their preconditions were met and were being considered for running at the same time. These behaviors are given the *activation potential* value which is high enough to always be larger than the *activation potential* of the *move* behaviors, which hit their maximum value at the closest distance before the behavior is assumed to be done. In order to run, the *pick* and *place* behaviors require shared control of the move base, exclusive control of a gripper, and shared control of the robot's head.

C. Task Nodes

To enable multitasking, we implemented a novel approach for the two-way activation spreading in the HTT. First, in the top-down activation process, we include the state of the available resources to all the messages being passed down from parent to children nodes. In this way, all of the nodes in the tree have an accurate representation of what is available when they compute their *activation potential*. After the activation spreading, in the bottom-up messages, each of the leaf nodes send a request for resources up the tree, along with their *activation potential* and whether or not their request is possible (i.e. whether or not the resources required are available). Then they wait for the request to be accepted or denied by the parent nodes before starting their work or sleeping, respectively. The main assumption that is made by the root node of the tree is that on the way back up the tree, each task node finds the locally most efficient request from its children to pass up. This results in only the compatible and maximally efficient set of requests in terms of *activation potential* reaching the root node. The root simply has to accept the request and allow the nodes whose requests reached the top to run while all other nodes go to sleep until the next time activation is spread from the root node, which happens when any of the resources are released.

In addition, there are other considerations that the task nodes have to address for multitasking, in particular under the **AND** and **THEN** nodes.

*1) **AND** Node:* From the perspective of multitasking, only nodes placed under an **AND** node can be ran in parallel, thus requiring a mechanism to find compatible behaviors that can be performed simultaneously. **OR** nodes activate a single child node and **THEN** nodes can only run their children in order, meaning one child node at a time. It is important to note, however, that any task node can pass up multiple parallel request in case there is some descendant **AND** node lower in the tree.

In this node, we designed a mechanism to find the compatible grouping of nodes to transmit up in the tree with the highest total *activation potential*. This decision is based on the requests received from lower-level nodes, as shown in Alg. 1. First, activation is spread to all the children nodes in order to receive their activation potentials and resource requests back. Next, after the requests are received they are flattened into a list of requests from one node each. If there is a descendant **AND** node, the requests that were passed up together will be given equal consideration as all other requests, rather than being considered together. For example, if a child behavior and a descendant behavior which is multiple levels of depth lower in the tree each have the highest *activation potential* and are compatible they should be passed up together. Next, the requests are sorted in descending order based on their *activation potential* and the first request which is possible to fulfill, i.e., all required resources are available, is added to the list of requests to pass higher in the tree because it is behavior with the highest perceived efficiency. After that, the **AND** node checks each subsequent request for compatibility with the list of accepted requests and if any are compatible with the current request they are added to the list. Finally, the compatible requests are simply passed up the tree and all other children are rejected.

*2) **THEN** Node:* For these task nodes there could be situations when the goals of children nodes are achieved through changes in the environment or through the robot's

Algorithm 1 Returns the list of compatible behaviors with the highest total activation potential to run together. r is the list of requests sorted by *activation potential*, and *cState* is the current state of the resources.

```
procedure $FindCompatible(r[1...n], cState)$
    $tRequest \leftarrow \phi, i \leftarrow -1$
    for $i \leftarrow 1...n$ do
        if $isPossible(r[i], cState)$ then
            $tRequest \leftarrow tRequest + \{r[i]\}$
            break
        end if
    end for

    for $j \leftarrow i...n$ do
        $state \leftarrow copy(cState)$
        $state \leftarrow updateState(tRequest, state)$
        if $isPossible(r[j], state)$ then
            $tRequest \leftarrow tRequests + \{r[j]\}$
        end if
    end for

    return $tRequest$
end procedure
```

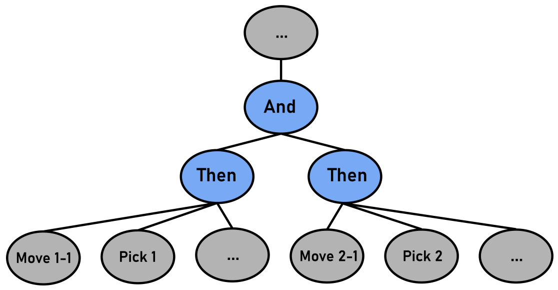

Fig. 3. A tree which both objects 1 and 2 need to be picked up. If both objects are in the same location, one of the two *move* behaviors should be skipped in order to ensure that the *pick* behaviors are run in parallel.

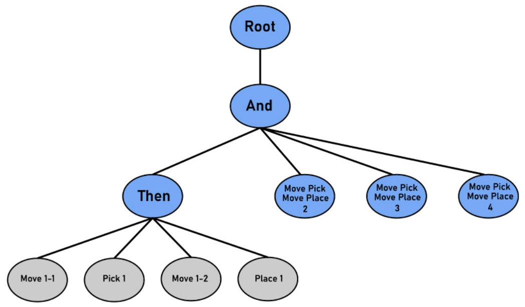

Fig. 4. The tree used to test the architecture. The Move-Pick-Move-Place nodes are actually represented by the subtree for object 1 on the left. They are depicted in shorthand to conserve space.

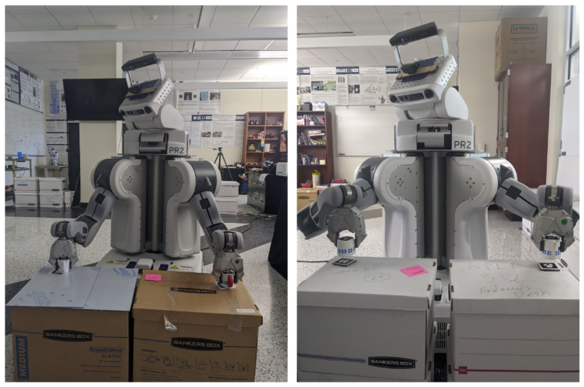

Fig. 5. These images show PR2 simultaneously running behaviors during experiment 1. The image on the left shows PR2 picking up both objects 1 and 2. The image on the right shows PR2 placing objects 3 and 4 together.

actions while in the process of attaining another goal. These situations should be detected and the architecture should prevent those nodes from being executed again.

A representative example is if there are multiple objects that need to be picked up from the same area, to prevent the robot from repeatedly moving to a space that it is already in it should instead skip redundant *move* behaviors, thus enabling compatible parallel requests to be conveyed upwards in the tree. The tree corresponding to this scenario is shown in Fig. 3. If the *move* behavior for object 2 is not skipped after the *move* behavior for object 1 then the candidate behaviors that are being considered by the **AND** node are: the *pick* behavior for object 1, and the *move* behavior for object 2. Since these behaviors are not compatible they will not be run together, however, intuitively the *move* behavior should be skipped for object 2 and the **AND** node can consider two *pick* behaviors instead, which are compatible. To address this situation, in the **THEN** node we added a process of detecting and skipping the first child node in the queue if it happens to have been fulfilled by an earlier completed task. Each behavior node continuously checks the state of the environment to see if its goals have been achieved. If they have been achieved, this information is passed up to the parent **THEN** node, which skips that behavior and looks for the request from the next child in the queue. If a behavior is skipped it is also not considered *done* until the subsequent behaviors are accepted and begin running. If they are at some point higher in the tree rejected, the **THEN** node reverts back to the first node which was skipped and resets its state for the next round of activation.

IV. EXPERIMENT DESIGN

To validate the architecture we designed and performed experiments using a PR2 humanoid robot, which is particularly well suited as it is equipped with two arms as well as a mobile base. The goal is to illustrate that the architecture enables the robot to multi-task (simultaneously pick up objects) while maintaining consistent allocations of resources to sub-tasks. For the experiments we employed a task that requires the robot to pick up 4 objects and place them in 4 different destinations, without a specific imposed ordering.

The task tree is shown in Fig. 4. We chose this tree because this is a fairly regular task for robots in any domain (e.g. tool retrieval, material collection, etc.). In the experiments, the initial location of the objects, their destination and the initial location of the robot are varied in four different configurations (Fig. 6), in order to illustrate the ability of the architecture to dynamically select, at run-time, the ordering of actions based on the state of the environment. For all the experiments, the robot has a map of the environment and knows the locations of the objects in the map. In this task the robot is both able to multitask as well as "split" pick-and-place tasks, illustrating a more efficient order of completion than a no-multitasking approach, as objects would have to be picked and placed sequentially, thus increasing the amount of time for the task. We chose specifically pick and place tasks because they can fully utilize the resources of the robot, however, this definition is generalizable to any tasks which follow our definition of low level behaviors.

For the purposes of evaluating the performance of the architecture, we consider optimal performance to represent cases in which actions that could be done simultaneously are scheduled in this way by the robot, as well as those in which all unnecessary behaviors are skipped. This can also be stated as the minimum number of time steps (where one step is one full action) a given robot can take to complete a given task. Optimality with respect to real time or optimal uses of resources could also be of use, however, these metrics are based also on the real world task parameters and therefore are hard to compare between our experiments.

V. RESULTS

To show the results of the experiment, we recorded timing diagrams of the active behaviors and their orderings. Each graph depicts all of the low level behaviors along the y axis and shows the times in which they were *waiting*, *active*, and finally *done* along the x axis. In each experiment, all tasks were completed successfully and the expected orderings were achieved as shown in Fig. 7. Below we give an explanation of the results and give a quick description of the task execution ordering chosen by the robot.

In experiment 1, the robot first moved to the pick destination for objects 1 and 2. The redundant *move* behavior was skipped and then both objects were picked up simultaneously.

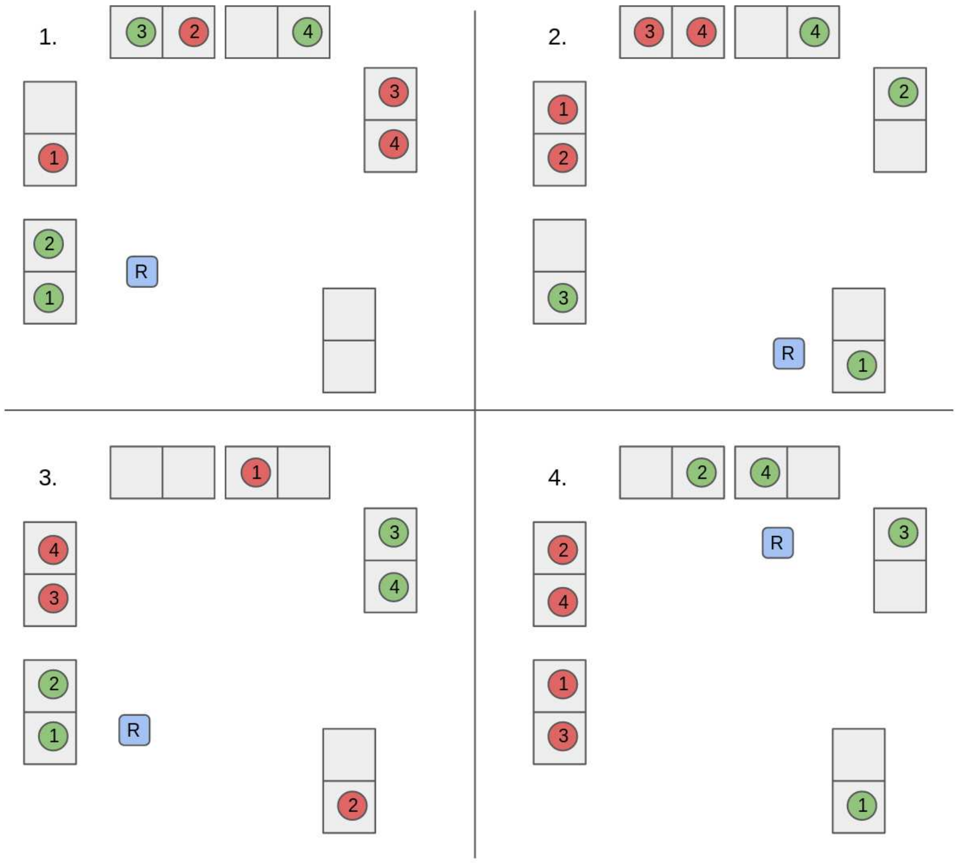

Fig. 6. The setup of the experiments used in this research. Each experiment in the diagram is numbered 1 through 4 on the top left. In each diagram the squares represent depots in which objects can be placed, the blue rectangle with the R represents the starting position of the robot, each numbered green circle represents where each respective object must be picked from, and each red circle represents where that objected must be placed.

Next, the robot moved to the place destination for object 1 and placed it. The robot then moved to the place destination for object 2. The *move* behavior for picking object 3 was skipped and object 2 was placed while object 3 was picked in parallel. Then, the robot moved to and picked up object 4 before moving to the final destination and placing objects 3 and 4 simultaneously. For this experiment, all behaviors that could be done simultaneously were performed at the same time. In addition, the robot was able to identify and skip all the behaviors that did not have to be executed, thus achieving optimal ordering based on our criteria. Fig. 5 shows the robot simultaneously picking objects 1 and 2 and placing objects 3 and 4 in this experiment.

In experiment 2, the robot moved to and picked object 1, followed by object 3. Then the robot moved to the place destination for object 1 and placed it. The same was done for object 3, after which the robot moved to and picked up object 4 and then object 2. Finally, those objects were placed at the destination one after the other in their respective destinations. This experiment was suboptimal, as there were no behaviors that were skipped or performed in parallel. However, time was still saved due to the fact that the *pick*-and-*place* behaviors were executed interspersed: objects were picked and/or placed along the way to other sources/destinations, which helped to minimize the distance traveled with the *move* behaviors.

In experiment 3, the robot first moved to the destination for picking up object 1. A redundant *move* behavior for object 2 was skipped and then objects 1 and 2 were picked simultaneously. Then, the robot moved to the destination for object 2 and placed it. The robot then moved to the pick destination for object 4 and picked it up, moved to the place destination for object 1 and placed it, and then moved back to the pick destination for object 3 and picked it up as well (objects 3 and 4 could be picked in the same place). Finally, the robot moved to the destination to place object 3, skipped the redundant *move* behavior for object 4 and placed both objects simultaneously. This ordering was efficient, as the unnecessary behaviors were skipped and two objects were placed simultaneously. However, a more optimal ordering would have also picked objects 3 and 4 up together.

In experiment 4, the robot moved to and picked objects 4 and 2 in sequence. Then, the robot moved to the destination to place object 2, skipped the redundant *move* behavior to place object 4, and placed both objects simultaneously. The robot then moved to and picked objects 3 and then 1 in sequence, and again moved to the place destination for object 3, skipped the redundant *move* behavior for object 1 and placed them both simultaneously. Due to the setup of the experiment this was the optimal ordering as the unnecessary

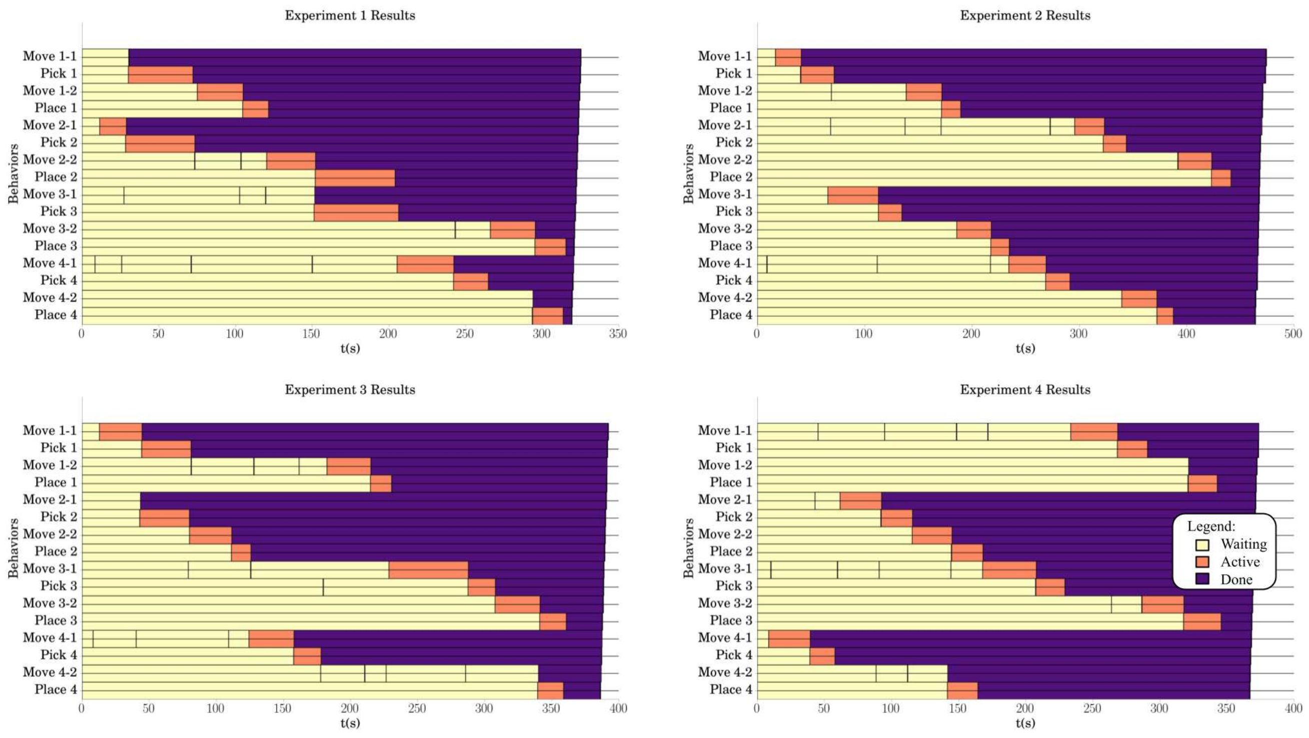

Fig. 7. Timing diagrams showing the results of all experiments. The x axis represents time in seconds, and the y axis represents the labeled behavior for each bar. The bar is yellow for *inactive* behaviors, orange for *active* behaviors, and *purple* for completed behaviors. The results show that behaviors can be skipped and can also be run simultaneously. For example, the Move 1-1 behavior in experiment 1 is skipped and the Pick 1 and Pick 2 behaviors in experiment 3 are run simultaneously.

move behaviors were skipped and all behaviors which could have been done in parallel were performed in parallel.

VI. DISCUSSION

Overall, the results of the four experiments showed both optimal and slightly sub-optimal results, in comparison with what we would consider the optimal orderings for completing the tasks. For each experiment the optimal order would have picked all objects that could be picked and/or placed together simultaneously, which would have also skipped the maximum number of behaviors. In experiments 1 and 4 the ordering that we observed was considered optimal based on how the experiments were set up. In experiments 2 and 3 we noticed some suboptimal orderings.

In experiment 3, we achieved a close result to the optimal ordering where there was just a single case in which the architecture could have decided to multitask but did not. This was when the robot first picked up object 4, then placed object 1, and then went back and picked up object 3. A more optimal ordering would have been to place object 1 first and then pick up objects 3 and 4 together, which would have skipped a *move* behavior as well. The reason that the sub optimal ordering was observed was because the *move* behavior to pick up object 4 had a higher *activation potential* than the *move* behavior for placing object 1. The computation of this potential is based on distance to objects, but could be further updated to incorporate other metrics.

In experiment 2 however, the architecture gained almost no advantage from the capabilities added to facilitate multi-tasking. The only advantage we gained in this scenario was that since behaviors were able to hold on to resources and were defined as single actions we did not have to serially perform each pick-and-place separately. This saved on the overall distance that the robot had to travel, but no behaviors were skipped and no behaviors were done concurrently even though some of them could have been. Again, the current implementation of the *activation potential* we used takes into account a distance-based metric, but other factors may be considered to address other aspects of efficiency.

However, the solution is not as simple as considering our conditions for optimality for each behavior's *activation potential* because those conditions require global information about the tree. Currently, the biggest advantage of the current implementation of the multitasking architecture is that computation is cheap and requires no explicit instruction to multitask. Therefore, any gain which we get from both the definition of low level behaviors or from concurrently running behaviors requires no additional considerations. While it does lead to suboptimal orderings with regards to our definition of optimality, generating optimal solutions is infeasible for arbitrary tasks.

In order to achieve the optimal orderings in experiments 2 and 3, some amount of global information about the tree is required to determine which behaviors can be skipped and which can be done in parallel ahead of time. However, each behavior and task node currently only has access to local information from itself and direct children. By limiting the decisions to be performed locally we make a greedy decision to looks at perceived efficiency (*activation potential*) rather than computing the optimal solution which is equivalent to solving the SPP. This is computationally infeasible. Therefore, our architecture provides a novel way of increasing the efficiency with which a task is completed by greedily assigning resources to tasks which as shown through our experiments can only make the task completion more efficient, and in some cases can provide optimal answers.

This architecture is opportunistic in nature with the intention of utilizing a greedy definition of perceived efficiency in order to pick what seems to be the best behavior to run at a given time and world state. As stated before, experiments 1 and 4 are encouraging examples of where this architecture can find optimal cases whereas in experiments 2 and 3 it performed similarly to a non-multitasking version of an HTT. It is important to note, however, that the worst case in terms of optimal orderings for this architecture is simply the sequential ordering which tries to minimize distance traveled. This highlights the benefits of allocating resources to tasks rather than robots to tasks. There is little overhead in terms of how long it takes for the architecture to find if behaviors are compatible and to maintain the status of the resources, and the benefits are that the orderings the architecture chooses will always be at least efficient with respect to the *activation potential* calculations. The results from experiments 2 and 3 show that the solution is not perfect. However, we did received benefit from tasks that can be performed concurrently and from tasks that can be skipped due to the changes implemented in this research.

In future work, we plan to consider resource management when deciding if tasks can be run in parallel: this serves as a suitable general solution for multitasking which can provide more efficient subtask orderings while still remaining robust to changing environments and unpredictable collaborators.

VII. CONCLUSION

In this paper we described an architecture based on HTTs that provides a general way for the robot to dynamically allocate its resources (sensors and actuators) in order to multitask, while ensuring consistency of the resource allocations. We achieved this by enabling the architecture to manage the availability of resources as well as the constraints imposed by the task. To support this and help create more efficient subtask orderings, we also enforced that low level behaviors are atomic modules that run completely uninterrupted in order to achieve a given goal. This enables the break down of more complex sub-tasks such as a *pick* and *place* behavior into low level behaviors that can be ordered dynamically and can also be performed concurrently alongside other behaviors, thus facilitating robot multitasking. We also maintained the overall efficiency of completing the task by utilizing both available resources and activation potential when considering which sub-task to run, and more importantly which subtasks are both compatible and efficient to run together. We then performed multiple experiments performed with a PR2 humanoid robot, showing that with the proposed approach to multitasking and managing resources the architecture provides benefit to the efficiency of completing any task by skipping redundant behaviors and performing compatible behaviors in parallel.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation (IIS-1757929, IIS 2150394).

REFERENCES

[1] R. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal on Robotics and Automation*, vol. 2, no. 1, pp. 14–23, March 1986.

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

[3] M. Colledanchise and L. Natale, "Handling concurrency in behavior trees," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2557–2576, 2022.

[4] ——, "Improving the parallel execution of behavior trees," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 7103–7110.

[5] J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel, "Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding," in *2010 IEEE International Conference on Robotics and Automation*, May 2010, pp. 2308–2315.

[6] K. Lakshmanan, A. Sachdev, Z. Xie, D. Berenson, K. Goldberg, and P. Abbeel, *A Constraint-Aware Motion Planning Algorithm for Robotic Folding of Clothes*. Heidelberg: Springer International Publishing, 2013, pp. 547–562.

[7] J. K. Behrens, R. Lange, and M. Mansouri, "A constraint programming approach to simultaneous task allocation and motion scheduling for industrial dual-arm manipulation tasks," in *2019 International Conference on Robotics and Automation (ICRA)*, ser. 2019 International Conference on Robotics and Automation (ICRA), 2019, pp. 8705–8711. [Online]. Available: https://doi.org/10.1109/ICRA.2019.8794022

[8] L. Jin and S. Li, "Distributed task allocation of multiple robots: A control perspective," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 5, pp. 693–701, 2018.

[9] 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: IEEE, November 2016, pp. 697–704.

[10] J. Blankenburg, S. B. Banisetty, S. P. Hoseini, L. Fraser, D. Feil-Seifer, M. Nicolescu, and M. Nicolescu, "A distributed control architecture for collaborative multi-robot task allocation," in *International Conference on Humanoid Robots*, Birmingham, UK, November 2017.

[11] B. A. Anima, J. Blankenburg, M. Zagainova, S. P. Hoseini, M. T. Chowdhury, D. Feil-Seifer, M. Nicolescu, and M. Nicolescu, "Collaborative human-robot hierarchical task execution with an activation spreading architecture," in *International Conference on Social Robotics*, Madrid, Spain, November 2019, pp. 301–310.

[12] J. Blankenburg, M. Zagainova, S. M. Simmons, G. Talavera, M. Nicolescu, and D. Feil-Seifer, "Human-robot collaboration and dialogue for fault recovery on hierarchical tasks," in *International Conference on Social Robotics (ICSR)*, CO, October 2020.

[13] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[14] M. R. Garey and D. S. Johnson, ""strong" np-completeness results: Motivation, examples, and implications," *J. ACM*, vol. 25, no. 3, p. 499–508, July 1978. [Online]. Available: https://doi.org/10.1145/322077.322090

[15] A. Atamturk, G. Nemhauser, and M. Savelsbergh, "A combined lagrangian, linear programming, and implication heuristic for large-scale set partitioning problems," *Journal of Heuristics*, vol. 1, pp. 247–259, 01 1996.

[16] K. L. Hoffman and M. Padberg, "Solving airline crew scheduling problems by branch-and-cut," *Manage. Sci.*, vol. 39, no. 6, p. 657–682, June 1993.