Task Assignment and Motion Planning for Bi-Manual Mobile Manipulation

Shantanu Thakar, Ariyan Kabir, Prahar M. Bhatt, Rishi K. Malhan, Pradeep Rajendran,
Brual C. Shah, and Satyandra K. Gupta†

***Abstract*— We present a two-layered architecture for task-agent assignment and motion planning of Bi-manual Mobile manipulators for executing complex tasks. We use search trees in temporal windows to determine feasible task assignments of agents using task and spatial constraint-based heuristics. We also introduce a caching scheme for moving between different trees so as to avoid re-planning of those portions of the robot motion that were successful. This greatly reduces the number of calls to the motion planner as compared to direct motion planning after task-agent assignment. We have shown our approach works on a set of complex tasks with significantly lower computation times.**

I. Introduction

In our daily lives, we perform many tasks which require us to use both our arms to manipulate a variety of objects. Many such tasks cannot be completed without coordination between the two arms. Moreover, some tasks also require us to manipulate objects while walking. Bi-manual mobile manipulators are becoming popular because they resemble humans in dexterity and can perform a variety of tasks which a single arm mobile manipulator may not be able to perform. For example, carrying large objects, assembling furnitures, opening and holding a CNC machine door while trying to extract the part etc. These tasks can be broken into subtasks. To successfully perform a task, the subtasks need to be assigned to the appropriate (left or right) manipulator.

Each arm of a bi-manual mobile manipulator needs to be assigned appropriately to carry out different subtasks. Moreover, the mobile base may need to assist either or both the arms in performing those tasks. A symbolic task-agent assignment without taking into account the feasibility of the motions of the agents may lead to failure in task execution. Consider a bi-manual mobile manipulator robot performing the task of attaching a board on the wall using a drill machine as shown in Fig. 1. We observe that a wrong assignment of the arms can result in infeasible (here in collision) final configuration of the robots.

Most of the Task and Motion Planning (TAMP) algorithms [1] can generate a task sequence for the agents involved and their corresponding trajectories. These algorithms invoke the motion planning routine for each task-agent assignment to validate the assignment. This approach is computationally viable when the query to the motion planning routine is not computationally intensive. However, in the case of a bi-manual mobile manipulator where for some tasks both the manipulators and the base need to work collaboratively, invoking motion planning for each agent assignment might be computationally expensive due to the high number of Degrees of Freedom (DOF). If the wrong arm is assigned for a task, the notification for this will come only after the motion planner returns an infeasible trajectory. This poses a challenge when we want to execute complex tasks using a bi-manual mobile manipulator.

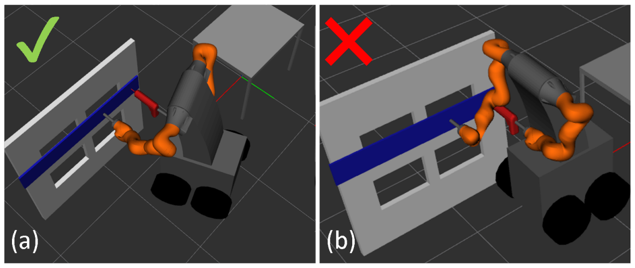

Fig. 1: A Bi-manual Mobile Manipulator (BMM) has to attach a part (blue board) to the wall using a drilling machine. The BMM has the agents (M_l, M_r, B) (right arm, left arm, mobile base). If it is assigned M_r for holding the part and M_l for drilling as shown in (b), it will result in an infeasible and in-collision pose for doing the task. Whereas the opposite assignment will result in a feasible pose without collisions for the task.

In this work, we have developed a search based algorithm for determining the correct task-agent assignment for generating feasible motions for a bi-manual mobile manipulator. We assume that the task network is given to us, which is independent of the robot. We propose spatial constraint based heuristics to prune branches of the search tree such that the motion planner is not executed for the wrong task-agent assignment. The developed algorithm selects the correct DOF needed to execute the given task. Moreover, we have a separate tree for motion planning once the task-agent assignment is done. We have introduced a caching scheme to move from motion planning to task-agent assignment if the motion plan fails and assign a different set of agents only for the part of the motion that failed. This helps in avoiding repeated calls to expensive motion planners and significantly reducing computation time.

II. Related Work

Robots like Herb 2.0 [2], HoLLiE [3], Rollin Justin [4] have been developed and studied for bi-manual mobile manipulation. In [5] parallelized planning and execution of tasks is implemented using Rollin Justin. Independent tasks are planned using parallel path planners and dependent tasks which need both the arms are planned using a single motion planner. Significant work has been done in the combined task and motion planning for such robotic systems. An interface between the task planner and the motion planner is developed in [6], which generates errors giving feedback to the task

†University of Southern California, CA USA
[guptask]@usc.edu

planner in terms of logical predicates when motion plans fail. Off-the-shelf task and motion planners are used along with geometric information, making them efficient. Many specialized search-based approaches for solving similar task and motion planning problems have been implemented [7]–[9]. However, in these methods, failing of the motion plan is the only notification for an incorrect task assignment. In [10] geometric backtracking is implemented for combined task and motion planning. This enables the robot to backtrack into the task planning domain if motions are not feasible.

Various heuristics have been suggested for combined task and motion planning methods [11]–[13]. Heuristics guide the search towards feasible motions thus reducing the number of motion plan attempts. In our approach, We assume that a task network is already given to us. We focus on the task-agent assignment and motion generation for a bi-manual mobile manipulator for the given task network such that motions are feasible and the task objectives are achieved.

In systems like bi-manual mobile manipulators, the motions of the mobile base and each of the arms is coupled. Task and agent assignment and scheduling for robotic systems have been studied in detail in [14]. However, in these methods, multiple robots with shared workspace do not move at the same time. Geometric conditions of manipulators are integrated with a symbolic description for task and manipulator motion planning in [15]. This enables performing manipulation tasks with several robots and objects. Combined task and motion planning is studied for a bi-manual humanoid setup in [16] focusing on perception and plan execution. Mobile manipulator task and motion planning have been implemented in [17]. Here, a hierarchical planner is presented where kinematic solutions are determined for task-level problems to determine optimal solutions for abstracted problems. Mobile base positioning is especially important in bi-manual setups, where the two arms can perform independent tasks if the mobile base is located at the appropriate position [18].

Point-to-point($p2p$) planning and constrained motion generation are both required for bi-manual mobile manipulators. There have been several approaches for manipulator motion planning like randomized algorithms [19]–[21], search based algorithms [22], [23], and optimization based methods [24]. For constrained motion generation for high degree of freedom systems like bi-manual mobile manipulators, we use sequential optimization described in [25], [26].

III. Problem Formulation

Definitions *Object:* Let, O_i be an object. In this paper, we will assume that O_i is a rigid body with a coordinate frame attached to it. We assume that the $state$ of O_i is defined as its the pose ($p = x, y, z, q_x, q_y, q_z, q_w$ or $p = x, y, z, \alpha, \beta, \gamma$) with respect to the world coordinate frame (W).

Bi-manual mobile manipulator: Let BMM refer to the bi-manual mobile manipulator into consideration. Let M_l and M_r refer the left and the right arms (serial-link manipulators) respectively and let B represent the mobile base of the robot. Here, each of the arms may or not have the same number of DOF. Without loss of generality, we consider two similar arms with n DOF $(\theta_1, \theta_2 \dots \theta_n)$. The mobile base configuration is represented by (x, y, ϕ) for its location and orientation. We assume each arm to be equipped with the appropriate end-effector to execute the desired tasks. We denote any of the arms or the base as an agent.

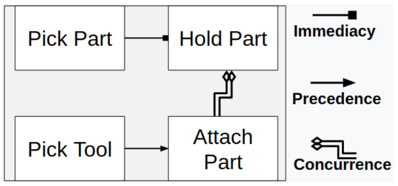

Fig. 2: Task network for the example shown in Fig. 1.

Task: Let T_k be a task. A task is defined by the state transition of an object. T_k, is associated with the pre-conditions (p_{pre}) and post-condition (p_{post}) of the state of one object. We assume that the associated object is available at its pre-condition state before the task starts. We assume that the type and number of agents required by the task are specified. One or more agents will carry out the task based on the requirement. The object will be transferred to the post-condition state after the agent/s have completed the task. We assume that the pre/post-conditions are given as sets of poses (i.e., $p_{pre} \in P_{pre}$, $p_{post} \in P_{post}$, $|P_{pre}| \geq 1$,$|P_{post}| \geq 1$). During task execution, we assume that a fixed-joint formation is required between an agent and an object whenever the agent operates on the object.

Task Network Model: Let $\mathcal{T}$ be a task-network. There are one or more connected tasks in $\mathcal{T}$ that completes an operation. We assume that any two tasks (T_A, T_B) in $\mathcal{T}$ can have one of the following four interdependencies-

(1) No interdependency

(2) Precedence: If T_A has precedence over T_B, then T_B cannot start until T_A has been finished. Based on the availability, same or different agent can be assigned for T_A and T_B

(3) Immediacy: If an immediacy connection comes from T_A to T_B, then T_B has to start immediately after T_A. The state of the object at the end of T_A will be the state of the object at the start of T_B. Based on the object's state transition requirement, it may not be possible to assign different agents for T_A and T_B.

(4) Concurrence: We define $container\ task$ and $embedded\ task$ to explain concurrence. Let $container\ task$ be the task that will have a longer duration and $embedded\ task$ as the task that is overlapped by the $container\ task$. Let, T_A and T_B two concurrent tasks. Let, T_A be the $container\ task$ task and T_B be the $embedded\ task$. By definition, T_B cannot start until T_A has started and T_A cannot be finished until T_B has finished. Therefore, the same agent cannot be assigned for the $embedded\ task$ of a $container\ task$.

We consider each of the task given in $\mathcal{T}$ as a $core\ task$. We assume that there can be $supporting\ tasks$ required to facilitate the core task. For a BMM we assume that the supporting task will always be done by the mobile base B. Example, if we assign M_r to pick up an object, B will assist when the object is unreachable for M_r. There may be cases where B has to assist both M_r and M_l to perform the task.

Let us consider the example shown in Fig. 1. The associated task-network for the BMM is illustrated in Fig. 2. There are four tasks in this task network. We want to assign agents to carry out these tasks and generate trajectories for the agents. As an example, we will summarize the attributes/requirements of the tasks below. From our definitions of $precedence$, $immediacy$, and $concurrence$, and the task requirements, we can see that HoldPart (HP) has to begin immediately after PickPart (PP). AttachPart (AP) can not start until PickTool (PT) has finished and HP has started. HP cannot finish until AP is finished.

TABLE I: Task Description

	Associated Object	**Required agent type**	**Pre-condition**	**Post Condition**
PickPart	Part	Manipulator	$p_{pre} = p^{part}_{initial}$	$p_{post} \in P^{part}_{picked-up}$
PickTool	Tool	Manipulator	$p_{pre} \in P^{tool}_{initial}$	$p_{post} \in P^{tool}_{picked-up}$
HoldPart	Part	Manipulator	$p_{pre} \in p^{part}_{picked-up}$	$p_{post} = P^{part}_{on\ wall}$
AttachPart	Tool	Manipulator	$p_{pre} \in P^{tool}_{picked-up}$	$p_{post} = p^{tool}_{on\ wall}$

Problem Statement For a given task network $\mathcal{T}$, we need to sequence the tasks to satisfy the constraints, assign agents to the tasks, and generate their trajectories to carry out the tasks using a BMM. The trajectories for the agents must be generated s.t the pre and post-conditions of each task in $\mathcal{T}$ are satisfied. Therefore, we need to generate continuous trajectories for the agents such that there is a smooth transition between the end of one task and start of the next.

Formally, given a task network $\mathcal{T}$ and a BMM, our objective is to assign tasks to the agents, sequence the tasks, and generate continuous configuration space trajectories for each agent such that there is no conflict in the assignment and the task is executed successfully.

IV. Approach

In our approach to solving the task-agent assignment and motion planning problems for a BMM, we consider a two-layered architecture. Each layer is divided into temporal windows. The division into temporal windows is based on the task order and constraints. In the first layer which is the *task-agent assignment* layer, we assign which task is to be done by which arm(s) of the BMM. In the second layer which is the *motion planning* layer, we execute the motion planners for the task assignments from the first layer.

Task-Agent Assignment Layer: In this layer, we assign appropriate agents for the tasks in each temporal window. We create a search tree from the root node to the leaf node for each temporal window. The root and leaf nodes are virtual nodes. Here, a node means a task-agent assignment. For a BMM, each task can be completed by one of the following 6 sets of agents: M_l, M_r, (M_l, B), (M_l, B), (M_l, M_r), (M_l, M_r, B). If in a temporal window, there are n tasks, the root node will have $n \times 6$ successors, i.e for each task, there are 6 possible agent combinations to execute the task. For each of those $n \times 6$ tasks, we have $(n-1) \times 6$ successors for the remaining tasks. The branching factor here is large, however a large number of the nodes will have task-agent assignments that will be infeasible due to the task constraints or spatial constraints. We have developed heuristics to identify such nodes and prune those branches before we run motion planners for executing the tasks.

The order of the task-agent node expansion is based on the fact that, the lesser the number of DOF required to complete the task, the lesser will be the computation overhead for motion planning. For example, if a BMM has to pick up an object from a table and it is reachable for one of the arms, we do not want to plan for all the DOF for the robot. Planning for just that arm is sufficient. Hence, for any node the successor task-agent assignments nodes are expanded in the order of increasing DOF as follows: (M_l), (M_r), (M_l, B), (M_l, B), (M_l, M_r), (M_l, M_r, B). For the sake of convenience, we fix this order. This acts as a branch guiding heuristic.

1) *Task Constraint Heuristics:* The first task constraint heuristic is based on the *concurrency* constraint between tasks. For *concurrent* tasks, if M_i is being used for the *embedded* task, we can prune the branch which has assignment of M_i and (M_i, B) for the container task. The second task constraint heuristic is based on the constraints from the task network provided. If a task T_A has been assigned to the arm M_i, and there is a task T_B such that there is the *immediacy* condition from T_A to T_B, then task T_B must be assigned to M_i, (M_i, B) or (M_i, M_j, B). Here, tasks T_A and T_B can be in different temporal windows.

2) *Spatial Constraint Heuristic:* This heuristic is based on the reachability and existence of collision-free inverse kinematics (IK) for the agents assigned to the task in a node. Tasks in which the object needs only one arm to manipulate and is reachable for arm without the motion of the mobile base can be done using M_l or M_r or (M_l, B) or (M_r, B), i.e either use the appropriate arm or an arm + base together. For some tasks, it might not be possible to complete the task with a single arm or two arms unless the base is moved. For such tasks, we can only use (M_l, B) or (M_r, B). For tasks which need both the arms and the object is reachable for both, we can use (M_l, M_r) or (M_l, M_r, B). And finally, for tasks which need both arms and for at least one of them the object is not reachable, (M_l, M_r, B) must be used. These combinations are determined by the spatial constraint heuristic which checks for the reachability of arms and existence of collision-free IK solutions for arm(s) + base combinations. The check for reachability and existence of feasible collision free IK for high DOF is computationally efficient and helps us to reduces calls to the expensive motion planning query for each task-agent assignment. The details for spatial constraint heuristic are discussed later.

Motion Planning Layer: In this layer, we take in as input, the branches from the root node of the first temporal window to the leaf node of the last temporal window in the *task-agent assignment* layer. Here, we adaptively select the motion planners and generate motion for the agents. Each node in each of the search tree in this layer is composed of task-agents assignments. If by sampling the key-frames (6 DOF poses) of the associated object, we can determine the constraints on the object. This, in turn, helps us determine whether a point-to-point ($p2p$) or constrained motion generation is required for the assigned agents to execute each task. In addition to generating motion of the agents at each node, our method generates motion for the transition between the end of a tree

Algorithm 1 AgentAssignmentAndMotionPlanning($\mathcal{T}$)

```
1: $R \leftarrow \{M_l, M_r, (M_l, B), (M_l, B), (M_l, M_r), (M_l, M_r, B)\}$
2: $\Theta^* \leftarrow empty$    \\Global container for Best Solution
3: $\Theta^*.cost \leftarrow \infty$
4: $t_{max} \leftarrow$ Maximum computation time
5: $W \leftarrow$ IdentifyTemporalWindows($\mathcal{T}$)
6: $T \leftarrow \{\}$    \\Container for Search Trees
7: for $i \in \{1, \ldots, |W|\}$ do
8:     $t_{i,1}, t_{i,2} \leftarrow$ InitializeTree($i, \mathcal{T}, W$)
9:     $T \leftarrow T \cup t_{i,1} \cup t_{i,2}$
10: end for
11: AssignTaskAgent($t_{1,1}, \mathcal{T}, R, W, T, t_{max}, \Theta^*$)
12: return $\Theta^*$
```

Fig. 3: Algorithm 1

in one window and the beginning of a tree in the next. An example of the two layers and temporal windows for the problem in Fig. 2 is shown in Fig. 4.

Caching Scheme: The search trees are labeled as $t_{(i,j)}$ with i being the temporal window index and j being the layer index. There are 2 layers and say m temporal windows. In a layer, if we find a branch from the root node to the leaf node for $t_{(i,j)}$, we move on to the tree $t_{(i,j+1)}$ which is in the next temporal window. In the first layer, if there is no feasible branch from the root to the leaf for $t_{(i,1)}$, we go to the tree $t_{(i-1,1)}$ of the previous temporal window and prune the current expanded branch to continue the search to the next branch. If we reach the leaf node of the tree $t_{(m,1)}$ of the last temporal window in the first layer, we expand the tree $t_{(1,2)}$ of the first temporal window of the second layer.

In the second layer, for trees in each temporal window, we generate motions for the BMM. However, if the motion planner fails to find a feasible motion in a tree $t_{(i,2)}$, we directly move to the tree $t_{(i,1)}$ of the same temporal window but in the first layer. Also, if the motion planner fails to find a feasible transition between leaf node of $t_{(i-1,2)}$ and root node of $t_{(i,2)}$, we move to the tree $t_{(i,1)}$. In the tree $t_{(i,1)}$, we prune the current branch and explore other branches. If we find a different feasible branch here, we move back to the tree $t_{(i,2)}$ motion planning layer in the same temporal window. If there is any inconsistency between the nodes of tree $t_{(i,1)}$ (current temporal window) and the tree $t_{(i+1,1)}$ (next temporal window) in the first layer due to different agent assignment, we move to the tree $t_{(i+1,1)}$ instead and resolve those inconsistencies by expanding the tree again. This continues till the entire first layer has a consistent agent assignment based on the task constraint heuristic. Then we move back to the tree $t_{(i,2)}$ which had infeasible motion, to begin with. This caching scheme helps in avoiding re-exploration of the same nodes in trees due to the generation of infeasible motions during motion planning.

Algorithms: We explain our algorithm using the example in Fig. 2. In line 1 of Algo. 1, a container R stores all the combination of agents possible for a BMM. The partially expanded search trees for both the layers in the example are shown in Fig. 4 . The function *AssignTaskAgent* in Algo. 2 shows the pseudo-code of the search for task-agent assignment. Since there are two tasks in the first temporal window, the root node will have 12 successors. Each of these 12 nodes will have 6 successors for the other task. In Fig. 4, we show the first correct task-agent assignment. We assume

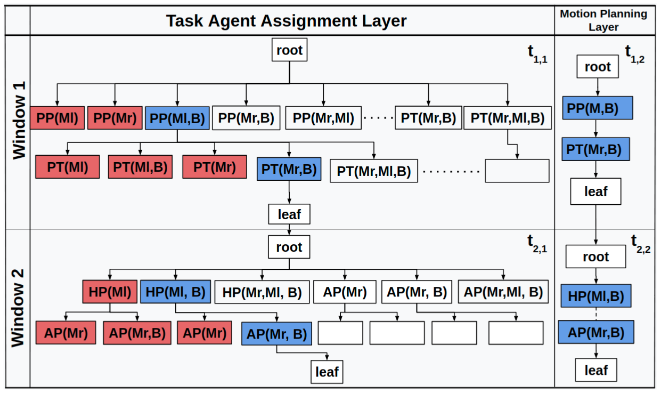

Fig. 4: Example of completely expanded search trees at different temporal windows of the three layers for the example in Fig. 2

that the initial location of the BMM is far away from the part and the tool. Hence, in $t_{(1,1)}$, the nodes which do not have the base B as an agent for the task will be pruned using the spatial constrain heuristic (marked red). Here, the node $PP(M_l, B)$ is selected for expansion. Once this node is expanded, a collision-free IK solution is generated for the agents combination at the pre and pose conditions of the object in the task and stored in the node.

The successor nodes for the $PP(M_l, B)$ node are shown. The nodes with the same agent like $PT(M_l)$ or same pair of agents $PT(M_l, B)$ will be still valid in here even though the same arm is being used for PP and PT. Those nodes will get pruned due to task constraint heuristics in the next temporal window due to the *concurrency* condition between the hold part (HP) and the attach part (AP) tasks. The nodes $PT(M_r)$ may or may not be pruned using the spatial constraint heuristic depending on the final configuration of the BMM at the parent node. If the arms are not reachable in $PT(M_l/M_r)$ and we do not find a feasible IK configuration for the nodes $PT(M_l/M_r, B)$, we prune those branches (using spatial constraint heuristic) and move on to node $PT(M_r, M_l, B)$. At this node, we find the IK for the whole BMM, to satisfy the post conditions of the task PT. Once we reach the leaf node of the tree in first temporal window, we move on to the root node of the tree in second temporal window. If the node $PT(M_r, M_l, B)$ has no feasible IK, we move to the $PT(M_r, B)$, and continue the search in the similar fashion. The function *isPruned* in line 8 of Algo. 2 executes these pruning rules. The *SelectNextTree* function in line 26 of Algo. 2 is used to select the next tree to move on to. It also encodes the caching scheme.

Spatial Constraint Checking: Nodes which assign a single arm to do a task, like $PT(M_l)$ in Fig. 4 are validated using the capability map of the manipulator. A capability map is used to perform collision-free reachability tests for a manipulator [18]. It is a pre-computed voxel grid, where a single query to each voxel returns if the centroid of the voxel is reachable by the robot in position and within a range of orientations. A single query on a pre-computed capability map is significantly faster than solving IK numerically.

We use analytical IK-solver for non-redundant manipulators and numerical IK-solver for redundant systems (i.e., mobile-base and arm/s combination) to test the collision-free

Algorithm 2 AssignTaskAgent($t_{i,j}, \mathcal{T}, R, W, T, t_{max}, \Theta^*$)

```
1:  while $ElapsedTime() \leq t_{max}$ do
2:      $node \leftarrow t_{i,j}$.PQ.pop()
3:      if not $node$.hasChildren() then
4:          break
5:      end if
6:      $S \leftarrow R$
7:      for $s \in S$ do
8:          if isPruned($s, \mathcal{T}$) then
9:              continue
10:         end if
11:         $t_{i,j}$.PQ.push($child, child$.cost)
12:     end for
13: end while
14: $T.update(t_{i,j})$
15: $flag, t_{i,j} \leftarrow$ IsBranchComplete($node, t_{i,j}$)
16: if not $flag$ and $i == 1$ and$j == 1$ then
17:     return
18: end if
19: $t_{fail} \leftarrow \phi$
20: if $flag$ and $i == |W|$ and $j == 2$ then
21:     $flag, \Theta, t_{fail} \leftarrow$ PlanMotion($T, \Theta^*$)
22: end if
23: if TrajExecutionTime($\Theta$) $<$ TrajExecutionTime($\Theta^*$) then
24:     $\Theta^* \leftarrow \Theta$
25: end if
26: $nextTree \leftarrow$ SelectNextTree($flag, t_{i,j}, T, t_{fail}$)
27: AssignTaskAgent($nextTree, \mathcal{T}, R, W, T, t_{max}, \Theta^*$)
```

Fig. 5: Algorithm 2

reachability at key-frames. If a task has to be performed by multiple manipulators, each attached to a different mobile-base, then we solve the IK for one manipulator/s and mobile-base combination at a time by keeping the configuration of others constant. The *CheckReachability* function in line 13 of Algorithm 3 checks these spatial constraints. Finding collision free numerical IK is computationally expensive as we have to include collision costs in the numerical optimization cost function [25]. Hence, we attempt to find the numerical IK without collision and check for collisions later. This is repeated a few times with different seeds for optimization. If there is no feasible solution from this method, we attempt numerical optimization with collision costs once. If this also results in no feasible IK solution, we prune that node.

Motion Generation: We have used sampling-based planners for the point-to-point motion of high-DOF systems, such as mobile base and arm combinations. We have used an optimization-based method to generate constrained trajectories. We pose the constrained trajectory generation problem as a discrete parameter optimization problem and solve it using successive refinement [25]. We set the objective to be minimizing the trajectory execution time and constraints to be (1) maintaining the constraints imposed by the task, (2) satisfying the kinematic and dynamic constraints of the robot, and (3) avoiding collisions. These motion planning methods are encoded in the *PlanMotion* function in line 21 of Algo. 2. This takes in the tree container T and the current best solution and finds if the next solution is of a lesser time cost. In this function, we make sure that the transition between the leaf node and the roots nodes of two successive trees in the motion planning layer is smooth. The t_{fail} output argument is to determine which tree of the motion planning layer has infeasible motions. This is useful to execute the caching scheme as explained in Sec. IV.

V. Results

We have implemented our algorithm for task-agent assignment and motion planning for 3 challenging tasks. The BMM we used has 17 DOF, 3 DOF for the mobile base and 7 for each of the arms. The arms are the KUKA iiwa 7 manipulators and mobile base is holonomic. The tasks executed by the BMM are described in Figs. 6, 7 and 8. There were 8, 5 and 12 sub-tasks in the task networks for the three tasks respectively.

To demonstrate the effectiveness of the spatial constraint heuristics and the caching scheme used, we show the time taken to find the first solution for our algorithm. Further, we compare it with direct motion planning without the spatial constraint heuristic and the caching scheme. The time taken for all the three tasks for both the cases is shown in Tab. II. We implemented our algorithm using MATLAB with an Intel Xeon 3.50GHz processor and 32GB of RAM. We set the total time limit for the algorithm to be 30 mins. For motion planners, we set the time limit to be 30s, 60s, and 90s for motions involving one, two, and three agents respectively.

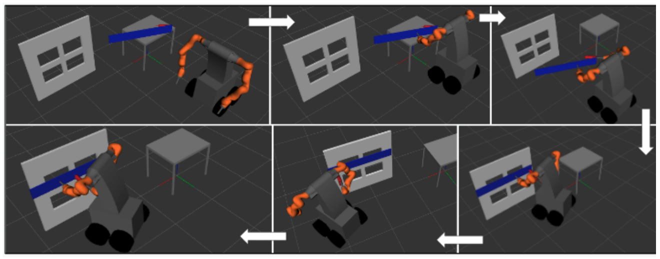
Fig. 6: Task No. 1 : The BMM has to pick up a long part and attach it to the wall using a drilling machine (blue). After it drills on the right end, it has to move and drill in the center while holding the part. It has to finally drill at the left end to complete the task.

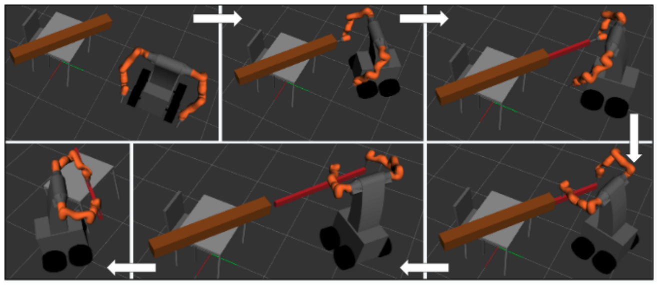
Fig. 7: Task No. 2 : The BMM has to take a long rod (red) out of a box (brown) and transfer it to a different location. It pulls out the part using the right arm, then holds it using both arms pulling it out completely. Thereafter, it transfers the part to a different location

TABLE II: Impact of having spatial constraint checking and caching in task-agent assignment and motion planning on computation time

Task No.	**With** spatial constraint checking and caching (sec)	**Without** spatial constraint checking and caching (sec)
1	32.3	479.7
2	15.3	257.5
3	71.9	921.3

It can be observed from Tab. II that with spatial checking and caching, on average the computation time for a solution is about 86% lower than without spatial constraint checking and caching. Where there is no spatial constraint checking, the motion planner is called for every task-agent assignment

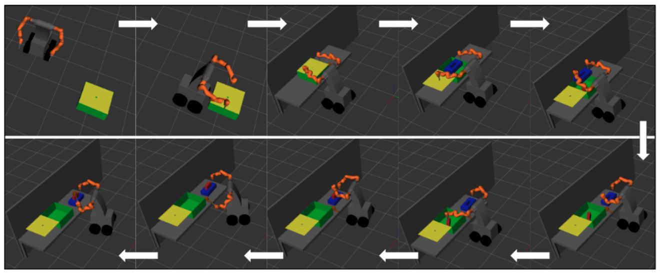
Fig. 8: Task No. 3 : The BMM has to pick up a box (green) and place it on the table, open it and pick up a part (blue) and place it outside. Then it has to pick up a small part (red) and assemble it with the blue part using a hammer

branch resulting in high computation time. Most of the initial branches are infeasible and that is known only after the planner attempts planning. The details of the task networks and the generated trajectories are available in the video at the following link https://youtu.be/MnghUBqQ6qg

VI. Conclusions

We have presented a two layered architecture for task-agent assignment and motion planning for a bi-manual mobile manipulator with a given task network. We have introduced spatial constraint checking heuristics for pruning branches of search trees in a computationally efficient way. This significantly reduces the computation time as the motion planner is not called at every task-agent assignment, but only those which are feasible in terms of reachability and collision-free IK solutions. Moreover, we also present a caching scheme in which we move between search trees when motion planner fails to find a solution for a task, so as to not plan again for the successful portion of the task. These two techniques significantly reduce the computation time. We have used point to point planners for the BMM and constrained motion generation wherever necessary. In the future we plan to expand this to a group of co-operating bi-manual mobile manipulators for even more complex tasks.

Acknowledgment: This work is supported in part by National Science Foundation Grant #1634431. Opinions expressed are those of the authors and do not necessarily reflect the views of the sponsor.

References

[1] F. Lagriffoul, N. Dantam, C. Garrett, A. Akbari, S. Srivastava, and L. Kavraki, "Platform-independent benchmarks for task and motion planning," *IEEE Robotics and Automation Letters*, vol. 3, pp. 3765–3772, Oct. 2018.

[2] S. S. Srinivasa, D. Berenson, M. Cakmak, A. Collet, M. R. Dogar, A. D. Dragan, R. A. Knepper, T. Niemueller, K. Strabala, M. V. Weghe *et al.*, "Herb 2.0: Lessons learned from developing a mobile manipulator for the home," *Proceedings of the IEEE*, vol. 100, no. 8, pp. 2410–2428, 2012.

[3] A. Hermann, J. Sun, Z. Xue, S. W. Ruehl, J. Oberländer, A. Rönnau, J. M. Zöllner, and R. Dillmann, "Hardware and software architecture of the bimanual mobile manipulation robot hollie and its actuated upper body," in *2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. IEEE, 2013, pp. 286–292.

[4] C. Borst, T. Wimbock, F. Schmidt, M. Fuchs, B. Brunner, F. Zacharias, P. R. Giordano, R. Konietschke, W. Sepp, S. Fuchs *et al.*, "Rollin'justin-mobile platform with variable base," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 1597–1598.

[5] F. Zacharias, D. Leidner, F. Schmidt, C. Borst, and G. Hirzinger, "Exploiting structure in two-armed manipulation tasks for humanoid robots," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 5446–5452.

[6] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *IEEE international conference on robotics and automation (ICRA)*, 2014, pp. 639–646.

[7] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical planning in the now," in *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[8] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.

[9] C. Reed Garrett, T. Lozano-Pérez, and L. Pack Kaelbling, "FFRob: Leveraging symbolic planning for efficient task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 104–136.

[10] J. Bidot, L. Karlsson, F. Lagriffoul, and A. Saffiotti, "Geometric backtracking for combined task and motion planning in robotic systems," *Artificial Intelligence*, vol. 247, pp. 229–265, jun 2017.

[11] R. Chitnis, D. Hadfield-Menell, A. Gupta, S. Srivastava, E. Groshev, C. Lin, and P. Abbeel, "Guided search for task and motion plans using learned heuristics," pp. 447–454.

[12] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Ffrob: An efficient heuristic for task and motion planning," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 179–195.

[13] A. Wells, N. Dantam, A. Shrivastava, and L. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *IEEE Robotics and Automation Letters*, 2019.

[14] S. Alatartsev, S. Stellmacher, F. Ortmeier, S. Alatartsev, S. Stellmacher, and · F. Ortmeier, "Robotic Task Sequencing Problem: A Survey," *Journal of Intelligent Robotic Systems*, vol. 80, pp. 279–298, 2015.

[15] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *The International Journal of Robotics Research*, vol. 28, no. 1, pp. 104–126, 2009.

[16] L. Karlsson, J. Bidot, F. Lagriffoul, A. Saffiotti, U. Hillenbrand, and F. Schmidt, "Combining task and path planning for a humanoid two-arm robotic system," in *Proceedings of tampra: Combining task and motion planning for real-world applications (icaps workshop)*. Citeseer, 2012, pp. 13–20.

[17] J. Wolfe, B. Marthi, and S. Russell, "Combined task and motion planning for mobile manipulation," in *Twentieth International Conference on Automated Planning and Scheduling*, 2010.

[18] R. K. Malhan, A. M. Kabir, B. C. Shah, and S. K. Gupta, "Identifying feasible workpiece placement with respect to redundant manipulator for complex manufacturing tasks," in *IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 2019.

[19] L. E. Kavraki, P. Svec, J.-P. Laumond, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration space," *IEEE Transactions on Robotics*, 1996.

[20] S. M. Lavalle and J. J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, 2001.

[21] P. Rajendran, S. Thakar, and S. K. Gupta, "User-guided path planning for redundant manipulators in highly constrained work environments," in *IEEE International Conference on Automation Science and Engineering (CASE)*, Vancouver, Canada, August 2019.

[22] B. J. Cohen, S. Chitta, and M. Likhachev, "Search-based planning for manipulation with motion primitives," in *International Conference on Robotics and Automation*, 2010.

[23] S. Thakar, L. Fang, B. C. Shah, and S. K. Gupta, "Towards time-optimal trajectory planning for pick-and-transport operation with a mobile manipulator," in *IEEE International Conference on Automation Science and Engineering (CASE)*, Munich, Germany, Aug 2018.

[24] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic Trajectory Optimization for Motion Planning," in *International Conference on Robotics and Automation*, 2011.

[25] A. Kabir, A. Kanyuck, R. K. Malhan, A. V. Shembekar, S. Thakar, B. C. Shah, and S. K. Gupta, "Generation of synchronized configuration space trajectories of multi-robot systems," in *IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 2019.

[26] S. Thakar, P. Rajendran, V. Annem, A. Kabir, and S. K. Gupta, "Accounting for part pose estimation uncertainties during trajectory generation for part pick-up using mobile manipulators," in *IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 2019.