Simultaneous task allocation and motion scheduling for complex tasks executed by multiple robots

Jan Kristof Behrens[1]   Karla Stepanova[1]   Robert Babuska[1,2]

***Abstract*— Coordination of multiple robots operating simultaneously in the same workspace requires the integration of task scheduling and motion planning. We focus on tasks in which the robot's actions are not necessarily confined to small volumes, but can also occupy a large time-varying portion of the workspace, such as in welding along a line or drilling a hole. Optimization of such tasks presents a considerable challenge mainly due to the fact that different variants of action execution exist, for instance, there can be multiple starting points of lines or closed curves, different filling patterns of areas, etc. We propose a generic and computationally efficient optimization method which is based on constraint programming. It takes into account the kinematics of the robot and guarantees that the motion trajectories of the robots are collision-free while minimizing the overall makespan. We evaluate our approach on several tasks of varying complexity: cutting, additive manufacturing, spot welding, inserting and tightening bolts, performed by a dual-arm robot. In terms of the makespan, the result is superior to task execution by one robot arm as well as by two arms not working simultaneously.**

***Index Terms*— task scheduling, dual-arm manipulation, motion planning, multi-robot systems,**

I. Introduction

In a multi-robot system, the individual robots are programmed to collectively perform a given task. Such a system not only can achieve goals that are infeasible for a single robot, but it also increases the overall performance thanks to the parallelization and combination of complementary robot capabilities. However, to fully leverage the robots' capabilities and to reduce their idle times, the individual tasks and motions must be properly coordinated. Coordination of multiple robots operating in the same workspace requires the integration of task scheduling and motion planing. Proper task scheduling determines the effectiveness and efficiency of the manufacturing process, while motion planning is necessary to compute collision-free plans for each of the robots. This is a non-trivial problem, given the huge state space spanned by the many degrees of freedom at task and motion planning level [1].

In the sequel, we refer to the integration of task scheduling and motion scheduling as task optimization. The tasks to be optimized can be divided into two categories: (i) tasks in which the robot's actions are confined to a small portion of the workspace (e.g., pick, place, apply glue to a point, etc.) – we call them confined tasks, and (ii) tasks occupying a larger volume which also varies in time (e.g., weld along a line, drill a hole, etc.) – we call them extended tasks. While the optimization of confined tasks has been sufficiently addressed in the literature [2], [3], extended tasks present a considerable challenge. This is due to time-varying space usage of the manipulator during action execution, and mainly due to the fact that different variants of action execution exist, for instance, there can be multiple starting points of lines or closed curves, different filling patterns of areas, etc. In this paper, we propose a constraint-optimization based method which is:

- generic, i.e., not restricted to a specific family of tasks and able to schedule both confined and extended tasks,
- computationally efficient and scales well with the number of tasks,
- guarantees that the motion trajectories planned are collision-free,
- can handle alternative, mutually exclusive task variants, e.g., multiple starting points or allocation to robot arms,
- enables easy definition of the task requirements as well as changes of the individual system parameters,
- takes into account the kinematics of the robot.

To find a collision-free plan which respects given constraints and minimizes the overall makespan, we use a constraint programming approach based on an extension of the STAAMS solver [3], so that it can handle tasks with time-

[1]Czech Technical University in Prague, Czech Institute of Informatics, Robotics, and Cybernetics, jan.kristof.behrens@cvut.cz, karla.stepanova@cvut.cz, [2]Delft University of Technology, Department of Cognitive Robotics, r.babuska@tudelft.nl.

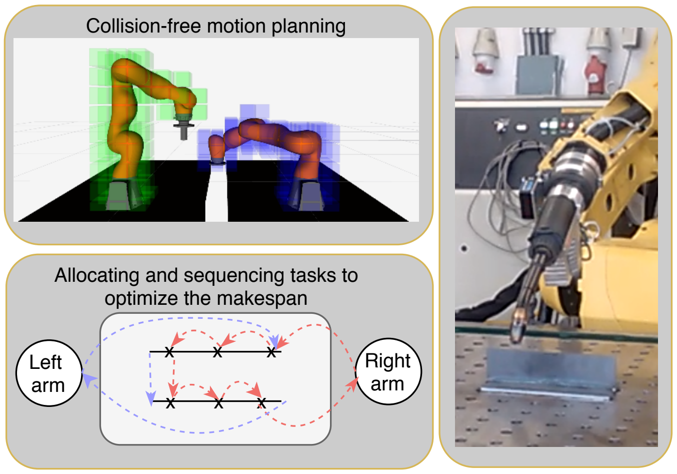
Collision-free motion planning
Allocating and sequencing tasks to optimize the makespan
Left arm
Right arm

Fig. 1. Scheduling tasks and motions for confined and extended tasks with a dual-arm robot. Top left: swept volumes are represented as voxels which allow for efficient collision checking and are naturally representable for the constraint solver used. Bottom left: task allocation and motion scheduling to optimize the makespan. Bottom right: Example of the welding machine moving a torch along a given trajectory to weld two objects.

dependent space occupation. Videos and additional materials can be found at the project webpage: `http://imitrob.ciirc.cvut.cz/schedulComplex.html`

II. Problem Definition

We address the following problem: Let A be a set of tasks to be executed, where the execution of task A_k during the time interval $[t_0, t_F]$ by a robot R_j leads to time and robot dependent occupancy of space: at each time interval $[t_i, t_{i+1}]$, where $i \in \{0, 1, ..., F-1\}$ a region $V_i \subset V$ is occupied by robot R_j. A solution to the problem is to find a sequence S of tasks along with their allocation to the individual robots R and collision-free motion plans for each of the robots with respect to the given constraints C, so that the overall execution time (makespan) is minimal.

III. Related Work

We first give a brief overview of the literature on sequencing complex tasks with volume occupancy. Then, we describe existing works on integrating sequencing of such tasks with motion planning, with a focus on robot-specific approaches.

Task sequencing. A thorough survey on robotic task sequencing is given in [1]. The sequencing of tasks which allow freedom in their execution order can be modeled as Traveling Salesman Problem with Neighborhoods (TSPN) [4]: given a set of polygons, the goal is to find a minimal-cost cyclic tour, such that it visits each polygon at its internal point. Alatertsev et al. [5] proposed a heuristic to optimize the sequencing of closed-contour robotic tasks and demonstrated the solution on a robot cutting holes in plastic. However, robot kinematics (motion planner) is not included into this approach. In [6], the authors deployed the solution on one KUKA arm and demonstrated how the production time decreases on test instances from the cutting-deburring domain. In [7], integrated task sequencing and motion planing for a welding robot was specified as an instance of TSPN problem. Also here, only one robotic arm was used. In [8], a solution to the robotic task sequencing problem is proposed which has a very low computational time. The authors applied their solution to the airbus shopfloor challenge and discussed the relation to TSPN.

The limitations of the above works are that they: (i) only address the sequencing of abstract actions and do not consider robot kinematics and motion planning; (ii) deal with one robot and do not address collision avoidance for motion planning of multiple robots; (iii) do not allow for constraints in the form of task subsequences with a fixed order.

Integrated task and motion planning. Although task and motion planning are often considered separately, in the case of multiple robots, it is important to plan both simultaneously. This entails decisions about *what* has to be done by which robot, *when* each subtask has to be performed and what is the concrete sequence of motions to realize. Integrated task and motion planning (ITAMP) is a subset of hybrid planning which deals with such problems. The difficulty of these planning problems stems from the fact that a task-level decision might be geometrically infeasible on the motion-level. The ratio of geometrically infeasible actions determines the best way to combine task and motion planning [9], [10]. Kimmel et al. [2] employ a time-scaling approach to schedule two given sequences of pick-and-place tasks. In [3], we compared simultaneous task allocation and motion scheduling approach for primitive tasks with pure time-scaling using an experimental setup similar to the one used by Kimmel et al. Akbari et al. [11] used ITAMP for a dual-arm robot in constrained table-top problems where the arms do not operate simultaneously. However, these planning problems try to generate sequences of actions which lead ultimately to the goal. In the industrial problems we are considering, it is by design known which actions have to be performed. The ITAMP approaches so far do not consider the optimization of cost criteria.

Constraint-based approach to task scheduling. The applications of constraint programming (CP) to multi-robot task planning and scheduling often use a simplified robot motion model and ignore the spatial interaction among robots in the scheduling process [12]. In this work, we employ CP to model the abstract task specification and the robot motion. Similarly, Ejenstam et al. [13] use CP to solve the problem of dual-arm manipulation planning and cell layout optimization via a coarse discretization of the workspace. Conversely, we create dense roadmaps to enable the close coordination of arms, thus allow simultaneous movements of arms. Kurosu et al. [14] describe a decoupled MILP-based approach to solve a simultaneous task allocation and motion planning, where the motion planner is prone to failure due to simplified motion and cost models used in the one-shot MILP formulation. This is not the case for us, as a single CP solver finds a mutually feasible solution for all sub-problems. In our previous work [3], we introduced a coherent formalism to model the robot and workspace as well as the abstract task plan and its invariants. We proposed ordered visiting constraints (OVCs) as task model primitives and time-scalable motion series as motion model primitives. In [15], we show how these tasks can be specified in a user-friendly manner by natural language and demonstration. However, only confined tasks were considered, without the freedom in action execution (e.g., selecting a starting point).

This paper's approach goes beyond the state-of-the-art in the works mentioned. The main contribution is that it performs simultaneous task and motion scheduling for multiple robots in the case of extended tasks including trajectory actions. It also handles constraints on the order of subtask sequences.

IV. Tasks and Motion Scheduling for extended tasks

Simultaneous task and motion scheduling (STAAMS) is concerned with the scheduling and allocation of high-level actions, while taking into account constraints at the motion level. Robot-robot collisions must be prevented, kinematics constraints and joint limits may not be violated. The result is a time-scaled trajectory for every robot arm that does not violate any constraints at the task and motion level during the task execution. The solver, see Fig. 2, is based on constraint programming (CP) and constraint optimization, sequentially solving constraint programming problems. A Constraint Satisfaction Problem (CSP) is generally specified by the triple (X, D, C), where X is a set of variables, D a set of domains, and C a set of constraints. The solution of a CSP is a complete assignment of values to variables that satisfies all constraints C. To find such a solution, the underlying solver performs a backtracking search over the variables with suitable variable and value selection heuristics. The search is interleaved with constraint propagation, which prunes the search by removing from the variables' domains those values that violate constraints in C. For the subsequent optimization of the makespan m_{end}, a series of CSPs with additional constraints on the makespan: $m_{\text{end}} \leq c_i$ (c_i being a current upper bound for the makespan), where $c_i < c_{i-1}$, is solved.

The STAAMS model [3] is organized in a task layer and a robot layer, see Fig.2, which extension for extended actions is described in the following subsections.

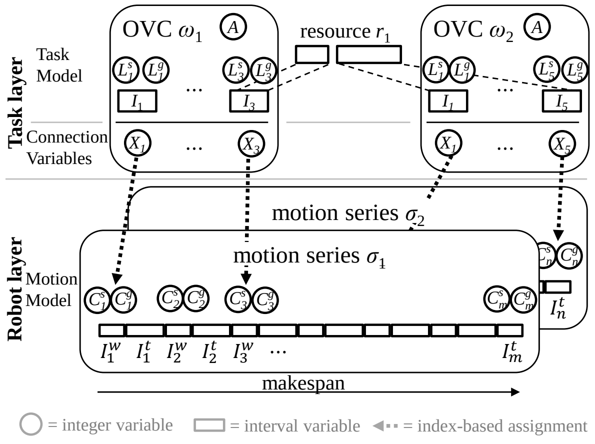

Fig. 2. CP model for task (task layer) and motion (robot layer) scheduling.

A. Task layer

The tasks are specified as a set of Ordered Visiting Constraints (OVC) with combinatorial and temporal constraints. Each OVC specifies a series of tasks which have to be executed by a single robot in a given order, e.g., a pick action followed by a place action, depositing of material along a trajectory followed by coating. In [3] these tasks were considered to only consist of very small movements with a near constant spatial footprint and controllable execution time. However, many relevant tasks violate this assumption. Consider, for instance, a welding seam: the robot positions the welding torch at the beginning of the seam, activates it and then tracks the seam at a constant speed. These extended tasks are characterized by the following properties:

- The start and ending configurations can be distinct and far apart.
- The volume occupied by the whole motion (swept volume) is large and the ratio of actually occupied volume to the swept volume is low.
- There might be multiple valid starting points for executing the task.
- The task duration can be long.

An extended task is defined by a list of possible starting locations L^s_j, corresponding goal locations L^g_j and trajectories T_j each represented by a list of k tuples (L_i, t^c_i), where L_i denotes the six degrees of freedom of the end-effector at time t^c_i). When the solver assigns the task to a given robotic arm and selects a starting location, then the motion plan of the robot is created and the corresponding robot configurations are assigned to each of these locations. The result is a list of k tuples (c_i, t^c_i), where c_i denotes the configuration robot visits at time t^c_i. These motion plans can be precomputed for each task variant in advance to save planning time.

Extended Ordered Visiting Constraints. Since extended tasks can be used interchangeably with confined tasks in OVCs, we add starting (L^s) and ending (L^g) locations in the OVCs (note that for confined action $L^s = L^g$) and corresponding robot configurations c^s and c^g to the motion series (see Fig. 2 top).

An OVC generalized for extended tasks is defined as the tuple

$$\omega = (A, [P_1, ..., P_l], [[L^s_1], ..., [L^s_l]], [[L^g_1], ..., [L^g_l]], [I_1, ..., I_l], [\text{cal}_1, ..., \text{cal}_l], C_{\text{intra}}). \quad (1)$$

An OVC ω models a sequence of confined and extended tasks to be executed at different locations L_i (6DoF of end-effector) by a given manipulator. A is a variable representing the *active component*, i.e. the manipulator, P_i defines the task type, e.g. apply glue, pick up an object or follow a line. These tasks have to be executed from an initial robot configuration corresponding to starting location L^s_i to a final robot configuration corresponding to location L^g_i. To execute the given task at the given starting location, a scripted task definition cal_i is called. This scripted task definition navigates the robot from the starting location L^s through the task, ending at a specified ending location L^g. These scripted task definitions can be defined by the user. The *time interval* variables I_i model the time windows for the task execution and C_{intra} is a set of constraints to model arbitrary relations between the OVC internal variables.

Extended tasks often can be fulfilled in one of multiple ways. Open contours can be started at either end and circular paths offer even more starting point choices. With the OVC constraint variables, defining these variants is very easy. An example for the line task to be executed by arm 'r_1' would be: ('make_line',[r_1],[L^s_1, L^s_2],[L^g_1, L^g_2]), where L^s_1 and L^s_2 are 2 alternative starting locations with corresponding goal locations L^g_1, L^g_2. For each of the possible variants, the corresponding robot motion plan is generated based on the predefined scripted task definition 'make_line'.

B. Robot layer

Industrial robots typically follow a workflow of alternating phases of effective and supporting movements [16]. In the case of multiple robots cooperating on the same task, this

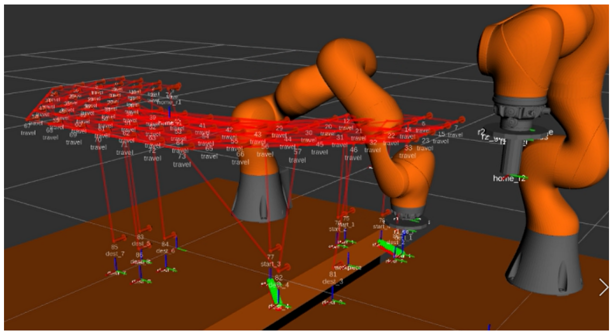
Fig. 3. Starting and ending locations of the extended tasks are connected to the closest nodes in the roadmap. This is done for the roadmaps of both arms (roadmap only for the left arm is displayed).

model does not provide enough flexibility, as the robots might block each other on their way to the next positions. Therefore, we introduce (a) waiting times after or instead of actions and (b) one intermediate configuration on the path to the next task location. This allows a time-scaling of the effective motions and evading movements during the support motions. The robot's motions are represented as a series of n joint configurations, n intervals modeling the time spent in these configurations, and $n-1$ intervals modeling the traveling time between the configurations (see Fig. 2 bottom). Precomputed roadmaps [17] are used to discretize the configuration space per arm (see Fig. 3). Therefore, the domain of the configuration variables is the set of all roadmap nodes of the corresponding arm. Path planning for supporting movements is performed by graph search on these roadmaps. In the case of the extended tasks when the occupancy of the space changes in time or when the starting and ending points are different, we have to allow connections of the starting and ending point of the task to different nodes of the roadmap, see Fig. 3.

C. *Collision-free plans*

Space representation for collision-free plans. Introduction of extended actions brings a huge number of robot configurations which have to be checked for collisions. To avoid extensive pair-wise collision checks, we utilize a voxel-based swept volume representation [18] for collision checking with precomputed swept volumes for each of the extended task trajectory. Additionally, we show how can be this space representation used that we do not have to reserve the whole swept volume (volume occupied by the whole motion) for the task duration, which is inefficient as it would prevent multiple robots from working efficiently in parallel.

Collision bodies of robots are typically represented as (triangular) meshes or compositions of primitive bodies like spheres, cylinders or cuboids. Although, optimized methods exist to check for the intersection of two bodies, the general case of checking two meshes against each other is computationally expensive. Since the solver is considering a large number of sequencing and task allocations, many robot configurations have to be checked for robot-robot collisions. As the STAAMS solver is implemented as a constraint program, the notion of resources is a natural way of modeling. Therefore, we divide the robot's workspace V into n volumes $v_i \subset V$ and treat the individual volumes v_i as unary resources. The volumes may not be overlapping, i.e., $\forall i,j : v_i \bigcap v_j = \emptyset$, and the whole workspace of the robot must be covered: $\bigcup\limits_{i \in 1 \dots n} v_i = V$. A voxelization of the volume V fulfills these requirements.

The space occupancy of a robot in configuration c_i can then be expressed as the set $V_i \subseteq \{v_1, \dots, v_n\}$. Figure 4 shows voxel occupancy for KUKA robot during a line movement. The space occupancy is piece-wise constant, i.e. V_i is constant from t_i to t_{i+1}. A new interval is added when the occupancy of the space changes. For efficiency reasons, this discretization can be limited by a temporal resolution or a maximum number, without sacrificing the safety of the execution. To account for kinodynamic constraints, we require that the robot always has enough free space in the direction of the movement to perform an emergency stop. The overall stopping time T_e is dependent on the velocity of the joints $\dot{x}_j$ and the acceleration limits $a_{\text{j,min}} \leq \ddot{x}_j \leq a_{\text{j,max}}$, $t_{\text{stop},j}$ denominates the minimal stopping time for a joint j: $T_e = \max_{j \in Joints} t_{\text{stop},j}$. Let T_e be the time to break to the full stop at time t_i. Then the volume to be reserved is $V_i^r = \bigcup\limits_{k \in K} V_k$, where $K = \{i, i+1, ..., j\}$, j is determined by the condition that: $t_j - t_i \geq T_{\text{stop}}$ (see Fig.5). We determined empirically the worst-case run-time of the stopping trajectory calculation $t_{\text{calc}} \leq 0.5$ s. The task execution has to be deterministic for in the sense that the actual motion is always contained in the reserved space V_i^r. Violations of this have to be handled by communication during execution (changed resource requirements). There is a trade-off between reserving the resources early and being able to move quicker and reserving less resources at the time.

To enable efficient use of resources, we free the space after a part of the task is fulfilled and the arm moves out of the given region.

Collision-checking To ensure collision-free motions of the arms, we determine the set of required resources and their timing relative to the respective action or traveling interval and cast temporal disjunctive constraints on potentially colliding motions. This ensures that intervals of the actions or supporting robotic movements which spatially overlap do not occur concurrently in the robotic plan. A fast collision check for two sets of voxels (vox$_1$ and vox$_2$) is done with the worst case runtime $\mathcal{O}(mn)$, where m and n are the sizes of the sets. For colliding sets the runtime is quicker as we can report collision after the first element found. For a voxel size of $10\,\text{cm}$ the occupied volume is represented by $80-90$ voxels. Empirically, the runtime was determined to 10^{-5}s. This sparse volume representation enables collision checks for arbitrary workspace sizes. As a result we get a collision-free motion plan (see Fig. 6).

V. Implementation and Setup

The proposed method is applied to a pair of simulated KUKA LBR iiwa robots with overlapping workspaces (see

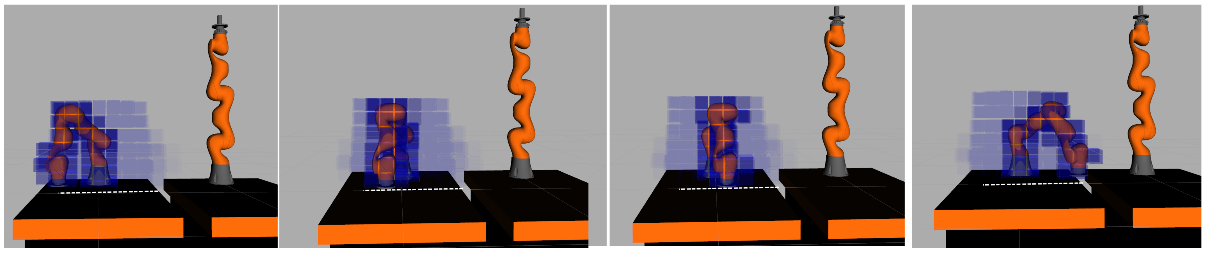

Fig. 4. Voxel occupancy during the line movement - light blue is the voxel occupancy for the whole trajectory (swept volume), dark blue is the occupancy for the current configuration - this is used for collision checking in the trajectory task computation (4 out of 45 configurations visualised).

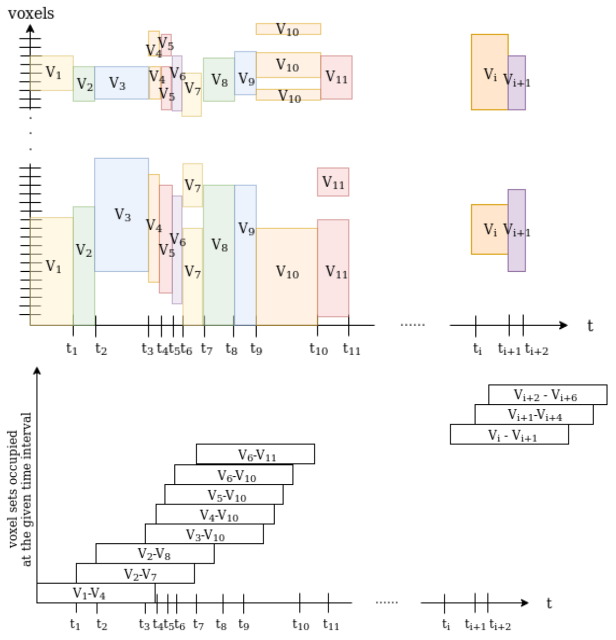

Fig. 5. (top) voxel occupancy at each time interval (V_i is the corresponding voxel set at the time interval $[t_i, t_{i+1}]$), (bottom) reserved and freed voxel sets are visualised at each time point (e.g. you can see that at time t_3, voxels from set V_2 are freed and voxels from voxel set V_9 and $V_1 0$ are newly reserved - resulting in the reservation of voxel sets V_3-$V_1 0$

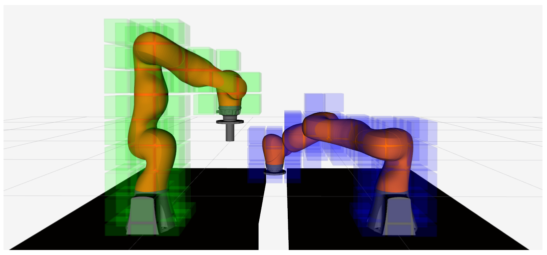

Fig. 6. Visualisation of voxel occupancy for individual robotic arms during the executed motion plan.

Fig. 6). The robots are controlled via the Robot Operating System (ROS). Motion planning is done with the MoveIt! Framework[19]. The roadmaps are based on the Graph-Tool library [20]. The STAAMS problems are translated into Constraint Programs which are solved by the Google Operations Research Tools Constraint Solver [21]. The STAAMS solver is accessible using ROS services. To create the voxelization, we arrange the meshes for the robot links according to the URDF model and the joint state of the robot. Then, we use the trimesh library [22] to retrieve the occupied voxels for the combined mesh. The evaluation was performed on an ASUS ZenBook Pro laptop with Intel i7-8750H CPU and 16 GB RAM. The solver runs in a single process on the CPU.

VI. Experimental Results

We evaluate our approach on multiple use-cases of varying complexity - cutting, additive manufacturing, spot welding and inserting and tightening bolts where multiple tasks by dual-arm robot have to be performed (see Fig.7). For cutting and welding, the robots have the same capabilities and for additive manufacturing and tightening bolts they do not. Extended actions are in some use-cases combined with confined actions (tightening bolts, spot welding). The makespan over planning time for individual found solutions is compared to the lower bound (not avoiding collisions). For each of the use-cases we compare multiple variants to show how the solver can handle the constraints imposed. These include comparisons to single arm performance (for tasks with shared capabilities), for arms not moving in parallel, enabling variants by starting point selection, automated allocation of tasks to individual arms, following/not following predefined order of tasks (e.g. first deposit material, then put coating).

Cutting showcases - (SC1) Making 8 line cuts (4 in the shared workspace), (SC2) making 11 line cuts (7 in the shared workspace). In Table I we compare results for the case where no alternative start location can be selected (No alt.) ws. when selection is enabled (Alt.start). We present a lower bound (Col.) to the optimal makespan where we omit the collision avoidance constraint. An upper bound is constructed by restricting the solver to only utilize a single arm. We can see that the overall makespan is improved by parallelization to 2 arms (No alt.) and further by utilizing the option of selecting a starting location for each line.

Additive manufacturing showcases (SC3) Material has to be deposited along a line, before coating is applied by the other robot, (SC4) material has to be deposited and coated along multiple lines. The starting location can be selected, but the order is fixed. We compare the cases when we allow collisions (Col.), where the arms cannot move simultaneously (No par.), and the case with free order (No order)

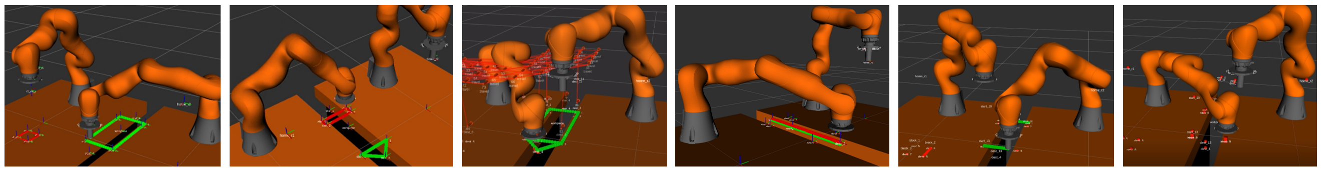

Fig. 7. Use-cases: (from left to right) 1) cutting example - multiple line cuts have to be performed; 2) cutting example in closer cooperation; 3) additive manufacturing - arm 1 is depositing material on multiple places and arm 2 is covering it with coating; 4) additive manufacturing - one arm is slowly depositing material along a long path, the other is coating it; 5) spot welding - arm 1 is making stitches on multiple places, arm 2 is making a weld seam; 6) bolt insertion - arm 1 is inserting bolts, arm 2 is tightening them(see accompanied video and project webpage).

TABLE I

CUTTING TASKS WITH ALTERNATIVE STARTING LOCATIONS

		Col.	1 arm	No alt	**Alt.start**
SC1	Final makespan [s]	20.7	45.5	28.3	**27.2**
	Time to 1st sol. [s]	0.14	1.3	2.4	**1.9**
	# solutions (1 min.)	5	5	2	**3**
	Time to final sol. [s]	1.47	74.45	25.4	**13.3**
SC2	Final makespan [s]	43.6	71.6	63.3	**48.8**
	Time to 1st sol. [s]	0.19	1.6	63.8	**108.3**
	# solutions (10 min.)	3	6	2	**4**
	Time to final sol. [s]	3.95	19.3	72.1	**283.7**

to the solution respecting the order of material application (Order). As can be seen, the planner is able to find a solution respecting the given conditions while improving the overall makespan compared to the case when two arms cannot move simultaneously.

TABLE II

ADDITIVE MANUFACTURING (GIVEN ORDER OF EXTENDED ACTIONS)

		Col.	No order	**Order**	No par.
SC3	Final makespan [s]	38.4	94.9	**91.4**	99.5
	Time to 1st sol. [s]	0.19	394	**108.3**	3.6
	# solutions (10 min.)	6	1	**6**	5
	Time to final sol. [s]	3.4	394	**520.7**	34.5
SC4	Final makespan [s]	15.8	22.0	**22.0**	28.4
	Time to 1st sol. [s]	0.055	3.8	**1.63**	1.58
	# solutions (10 min.)	2	1	**3**	1
	Time to final sol. [s]	0.055	3.8	**10.55**	1.58

Showcases combining confined and extended tasks (SC5) Spot welding - first stitches have to be done, then welding line can be applied, (SC6) Bolts insertion - 1 arm is inserting bolts, 2nd arm is afterwards tightening them. The performance is compared to allowed collisions (Col.) and the case when the ordering constraint is omitted.

TABLE III

TASKS COMBINING CONFINED AND EXTENDED ACTIONS.

		Col.	No order	**Order**
SC5	Final makespan [s]	53.9	89.2	**89.4**
	Time to 1st sol. [s]	0.28	273	**12**
	# solutions (10 min.)	5	1	**3**
	Time to final sol. [s]	3.1	273.1	**36.6**
SC6	Final makespan [s]	60.0	61.8	**71.4**
	Time to 1st sol. [s]	0.19	1.75	**1.76**
	# solutions (10 min.)	4	3	**3**
	Time to final sol. [s]	0.38	32.03	**33.91**

Makespan was reduced compared to the initial solution within 10 minutes for SC1, SC2, SC3, SC4, SC5 and SC6 by 10%, 46%, 7%, %, 9%, and %, respectively.

VII. CONCLUSION AND DISCUSSION

In this paper, we have introduced, demonstrated and evaluated an approach to simultaneous task scheduling and motion planning in multi-robot systems. It is based extensions of the CP-based STAAMS solver to support tasks which occupy a large, time-varying volume of the workspace and have multiple alternative starting points. We have shown that the system is able to schedule diverse tasks, e.g., cutting, additive manufacturing, spot welding, assembly, for robots tightly cooperating in a common workspace with many potential collisions. The proposed voxel-based space representation enables a very efficient collision check. This makes it feasible to check and schedule the effective motions (for both confined and extended tasks) as well as the supporting motions (travelling of robots via the roadmap edges) in a fine-grained manner.

We evaluated the proposed solution on several use-cases, showing that the makespan quality outperforms task execution by one robot arm as well as by two arms that cannot work simultaneously. In addition, within 2 minutes, the planner is able to improve the makespan of an initial solution by up to 46%.

The extensions presented widen the applicability of the solver. In or future research we will investigate how the quality of solutions depends on the possible supporting movements using the roadmaps. The coarse voxelization of $10\,\mathrm{cm}$ seems to prevent some potential for robot collaboration. Here, a trade-off between possible solution quality and planning time has to be made. We feel that there is a better Pareto optimum (without risking collisions and task fulfillment) to reach the rapid performance, we reported in our previous works [3], [15]. However, this space representation allows us to develop an execution engine with space reservation and monitoring. It also allows the efficient integration of other agents and objects occupying space at execution time.

VIII. ACKNOWLEDGMENT

This work was supported by the European Regional Development Fund under project Robotics for Industry 4.0 (reg. no. $CZ.02.1.01/0.0/0.0/15\_003/0000470$) and TAČR Zeta project Imitation learning supported by language for industrial robots (no. TJ01000470) granted by Technological agency of Czech republic.

References

[1] S. Alatartsev, S. Stellmacher, and F. Ortmeier, "Robotic task sequencing problem: A survey," *Journal of intelligent & robotic systems*, vol. 80, no. 2, pp. 279–298, 2015.

[2] A. Kimmel and K. E. Bekris, "Scheduling Pick-and-Place Tasks for Dual-arm Manipulators using Incremental Search on Coordination Diagrams," in *Proc. of PlanRob Workshop at ICAPS '16*, London, UK, June 2016.

[3] 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)*, May 2019, pp. 8705–8711.

[4] E. M. Arkin and R. Hassin, "Approximation algorithms for the geometric covering salesman problem," *Discrete Applied Mathematics*, vol. 55, no. 3, pp. 197–218, 1994.

[5] S. Alatartsev, V. Mersheeva, M. Augustine, and F. Ortmeier, "On optimizing a sequence of robotic tasks," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 217–223.

[6] S. Alatartsev and F. Ortmeier, "Improving the sequence of robotic tasks with freedom of execution," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 4503–4510.

[7] A. Kovács, "Integrated task sequencing and path planning for robotic remote laser welding," *International Journal of Production Research*, vol. 54, no. 4, pp. 1210–1224, 2016.

[8] F. Suárez-Ruiz, T. S. Lembono, and Q.-C. Pham, "Robotsp–a fast solution to the robotic task sequencing problem," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1611–1616.

[9] 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, 2017.

[10] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson, "Efficiently combining task and motion planning using geometric constraints," *The International Journal of Robotics Research*, vol. 33, no. 14, pp. 1726–1747, 2014.

[11] A. Akbari, F. Lagriffoul, and J. Rosell, "Combined heuristic task and motion planning for bi-manual robots," *Autonomous Robots*, vol. 43, no. 6, pp. 1575–1590, 2019.

[12] K. E. Booth, G. Nejat, and J. C. Beck, "A constraint programming approach to multi-robot task allocation and scheduling in retirement homes," in *International conference on principles and practice of constraint programming*. Springer, 2016, pp. 539–555.

[13] J. Ejenstam, "Implementing a time optimal task sequence for robot assembly using constraint programming," 2014.

[14] J. Kurosu, A. Yorozu, and M. Takahashi, "Simultaneous dual-arm motion planning for minimizing operation time," *Applied Sciences*, vol. 7, no. 12, p. 1210, 2017.

[15] J. K. Behrens, K. Stepanova, R. Lange, and R. Skoviera, "Specifying dual-arm robot planning problems through natural language and demonstration," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2622–2629, 2019.

[16] S. Alatartsev, M. Augustine, and F. Ortmeier, "Constricting insertion heuristic for traveling salesman problem with neighborhoods," in *Twenty-Third International Conference on Automated Planning and Scheduling*, 2013.

[17] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[18] A. Gaschler, R. P. Petrick, M. Giuliani, M. Rickert, and A. Knoll, "Kvp: A knowledge of volumes approach to robot task planning," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 202–208.

[19] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ros topics]," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[20] T. P. Peixoto, "The graph-tool python library," *figshare*, 2014. [Online]. Available: http://figshare.com/articles/graph_tool/1164194

[21] Google's or-tools. Google. [Online]. Available: https://developers.google.com/optimization/

[22] Dawson-Haggerty et al., "trimesh." [Online]. Available: https://trimsh.org/