Task Planning with Mixed-Integer Programming for Multiple Cooking Task Using dual-arm Robot

June-sup Yi[1], Min Sung Ahn[2], Hosik Chae[2], Hyunwoo Nam[2], Donghun Noh[2], Dennis Hong[2], Hyungpil Moon*

***Abstract*— This work proposes a task scheduling method in an optimization framework with applications on a dual-arm cooking robot in a controlled cooking environment. A mixed-integer programming (MIP) framework is used to find an optimal sequence of tasks to be done for each arm. The optimization is fast and simple as a priori information about the tasks to be scheduled reveal dependency and kinematic constraints between them which significantly reduces the problem size as infeasible solutions are removed pre-optimization. The optimization approach's feasibility is validated on a series of simulations and an in-depth scalability analysis is conducted by changing the number of tasks to be done, the dishes to be completed, as well as the locations where the tasks can be done. Considering the unique configuration of the platform, analysis on selecting the minimum time required tasks as opposed tasks that will give the most flexibility to the other arm is also done. An example is presented on a real set of tasks to show the optimality of the solution.**

I. INTRODUCTION

Traditionally, robotics have been successful in conducting a limited set of tasks in a known, controlled environment such as an assembly line or research laboratories. However, the number of robots coming out of such environments are growing by the day, with their purpose to automate many of the mundane tasks that people are forced to do.

The most successful types have been mobile robots, where iRobot's Roomba [1] series of vacuum cleaning robots are the epitome. There are also an increasing number of service robots that can guide people in pre-defined environments or assist people in carrying objects. More recently, we are also witnessing an increase of robots with limbs (i.e. manipulators and legged mobile platforms) which can do conduct much more tasks because of their relatively closer resemblance to a human's anatomy [2]. However, despite the relative increase in task versatility compared to their mobile counterparts, these limbed robots are still not openly available because of a combination of the maturity of the technology in an unstructured environment, the need for it in the general public, and the price point [3] [4].

*This research was supported by Woowa Brothers Corporation.

*This research was supported by the MOTIE(Ministry of Trade, Industry, and Energy) in Korea, under the Fostering Global Talents for Innovative Growth Program(P0008746) supervised by the Korea Institute for Advancement of Technology (KIAT).

[1]Sungkyunkwan University, 2066 seobu-ro, Suwon-si, Jangan-gu, Gyeonggi-do, Republic of Korea caro33@skku.edu

[2]University of California, Los Angeles, CA 90025, USA

*Corresponding Author, Sungkyunkwan University, 2066 seobu-ro, Suwon-si, Jangan-gu, Gyeonggi-do, Republic of Korea hyungpil@skku.edu

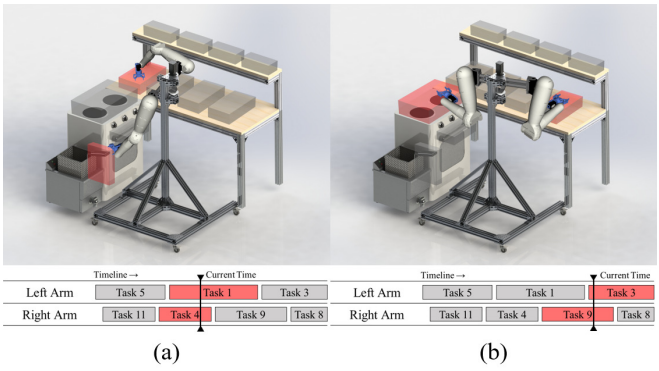

Fig. 1. The environment of the robot and the process of concurrently conducting different tasks with the two arms. (a) The robot conducting Task 1 and Task 4 at the same time. (b) After time has elapsed, the robot moves on to conduct Task 9 with Task 3 starting shortly after.

The success of these limbed robots in controlled environments is still encouraging, as it suggests that if the environment is somewhat controllable, many tasks that humans have to repeatedly do, which are too complex for mobile robots but can be done by one with arms or legs, could be automated. One of them is cooking. If robots could cook, they could possibly be the next Roomba as the kitchen environment is not drastically changing and because, similar to cleaning, it can be considered a mundane, repetitive task that alleviates a potential burden to young people [5]. This belief is supported by the relevant research and types of products that are being published and showcased by researchers and companies.

For example, Miso Robotics' [6] is selling manipulators that can flip burgers or work a fryer in a dedicated environment, successfully assisting human cooks with serving more than 15,000 burgers and 31,000 lbs of chicken tenders in 2019 alone [7]. Samsung's Bot Chef is able to prepare ingredients, cook on a frying pan, grab sauces out of cabinets, and generally assist another human with cooking [8]. LG's CLOi Kitchen Zone has a group of robots that can take orders to cooking bowls of noodles and making coffee for customers [9]. The general idea of automated cooking or robot-assisted cooking is well-represented, but in these demonstrations, a robot is given a single task to conduct at a time, which is contrasting to what humans might consider "optimal." Human experience tells us that we concurrently conduct tasks to minimize cooking time while still producing the same output quality.

Cooking is a combination of planning, multi-tasking, and

ultimately using prospective memory to complete sub-goals as well as the overall goal within a timeframe [10]. This results in a meal that can be delivered quickly. To the authors' best knowledge, such concurrency and multi-tasking exists at a minimal amount with the current culinary-related robots. There are works such as [11] which attempts to find an optimal starting time for cooking prior to delivery to maximize the "freshness" of the dish. On a broader scale in terms of task scheduling for robots, there are mixed-integer and constraint programs that schedule tasks for mobile robots, where each mobile robot has the full capacity to perform all tasks [12] [13]. However, for a pair of stationary manipulators, a combination of coupled joints between the manipulators or the potential interference in the workspace introduces additional complexities for the existing solutions to be viable. There was also a study applying Hierarchical Quadratic Programming (HQP) considering the limitations from robot configuration such as joint limit and singularity [14]. Our research differs in reducing the solving time by reducing the size of decision variables through preprocessing.

This work attempts to take a pioneering step in multi-tasking for a cooking robot such that the previously mentioned components that make up cooking can be achieved to ultimately minimize time while completing all the required cooking tasks. Furthermore, unique kinematic constraints are considered in the task selection, as the manipulators used have less than 6 DOF (5 DOF and redundant) for economic and kinematic advantages, which actually impose additional difficulties in the scheduling. To tackle this unique problem, an optimization framework is formulated using mixed-integer programming to assign each arm of a two-armed robot to a certain task at a given timeslot. A mixed-integer program is formulated to ensure all tasks are completed by being assigned a timeslot. Because the manipulators can concurrently conduct tasks at an order which optimizes for both time and task flexibility, the robot can complete all the tasks considerably faster than had it given its full attention to each task in a "blocking" fashion. Multiple tests are conducted on a different number of cooking scenarios with varying number of dishes and tasks to complete at random locations to show the effectiveness and the scalability of the approach.

Consequently, the main contributions of this work are the following:

1) Formulation of an optimization problem that assigns N (two in this work) manipulators discrete tasks at each timeslot using mixed-integer programming.
2) Scalability analysis of the optimization problem which reveals a relationship between the number of tasks, dishes, and task locations when optimizing for time and/or task flexibility, which results in a suggested combination of tasks, dishes, and task locations.

This work is organized as follows. We initially pre-process the solutions for the problem in Section II by eliminating options that are certain to not be the solutions. The entire framework is then explained in Section III, with the approach validated in Section IV. Section V concludes the paper while introducing some future work.

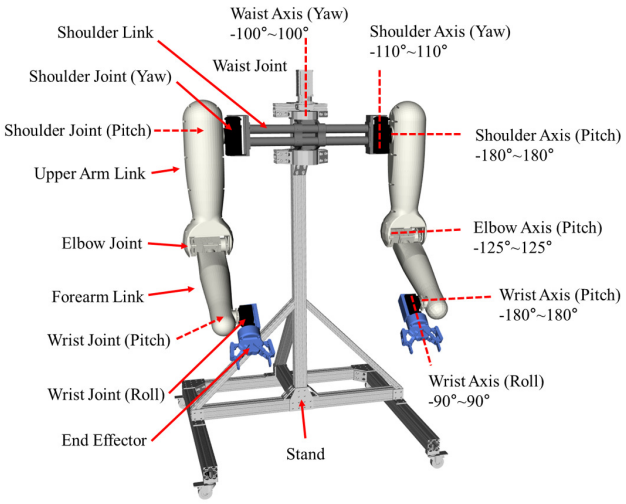

Fig. 2. 3D CAD model of the dual-armed robot

II. Problem Preparation

This section introduces the environment that the task scheduling problem is applied to and inspects its circumstances which sheds insight into areas where pre-processing of the ensuing optimization can be done.

A. Environment

The environment consists of an 11 degrees of freedom (DOF) dual-armed robot, where there are 5 DOF per arm with another revolute joint that couples the two arms at the center of the body. Despite the redundancy of each arm, becasue the arm is fully capable of pitch and roll motions, it is a sufficient platform for cooking in a fixed area. In the case that bi-manipulation is required, the coupling joint at the body can be used. Around the dual-armed robot are shelves on three sides with cooking appliances and ingredients ready for use. The whole environment is shown in Fig. 1, and a detailed 3D CAD Model of dual-armed robot and the angle range of each joints are shown Fig. 2.

B. Problem Reduction

Prior to formulating the optimization problem that would output tasks that are to be done by each arm, an analysis of the environment and the problem at hand can help embed good heuristics into the optimization such that the problem's complexity can be reduced. This is similar to how infeasible solutions can be eliminated with good heuristics and in essence, can be compared to tightening the bounds of a constraint.

1) Task Dependency Constraint: In cooking especially, order of operation is important. This means that there are tasks that can only be executed after successfully completing a preceding task. Therefore, when optimizing for a certain set of tasks to be completed by each arm, a constraint that

TABLE I

AN EXAMPLE OF A TASKSET.

$\mathcal{T}$ **No.**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
$\mathcal{D}$ No.	1	1	1	1	1	1	1	2	2	2	2	2
t	10	10	43	10	60	60	13	3	10	10	10	10
$\mathcal{L}$	8	10	4	8	4	1	5	6	10	11	3	9
d No.	-	1	2	3	4	5	6	-	8	9	10	11
$\mathcal{T}$ **No.**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	
$\mathcal{D}$ No.	2	2	2	2	2	3	3	3	3	3	3	
t	20	10	10	19	3	3	3	10	10	145	30	
$\mathcal{L}$	2	3	7	3	5	8	10	3	7	3	5	
d No.	12	13	14	15	16	-	18	19	20	21	22	

enforces dependency on the conclusion of another task is necessary.

To effectively represent multiple characteristics of each task, a hierarchical set called a $taskset$ is defined.

Definition A *taskset* T is an $\mathcal{T}$ array of quadruples consisting of $\mathcal{D}_i$, t_i, $\mathcal{L}_i$, and d_i, where $\mathcal{T}$ is the task, $\mathcal{D}$ is the dish, t is the time required to complete $\mathcal{T}$, $\mathcal{L}$ is the location where the $\mathcal{T}$ can be done, and d is $\mathcal{T}_j$ that needs to be completed before $\mathcal{T}_i$ can be started where $i! = j$. Note that all elements of the quadruple are required except for d.

To elucidate what may seem to be an abstract idea, an example is shown in Table I. The taskset in the example is information that is assumed to be available as in essence, it is a tabular representation of the time, steps, and location of tools required to prepare three dishes. For example, to prepare dish #3 ($\mathcal{D}$ = 3), tasks 18, 19, 20, 21, 22, and 23 must be done in that order.

Going back to the task dependency constraint, without optimization, it is clear that to successfully complete the preparation of the three dishes, the first task that the robot does must be either $\mathcal{D} = 1||2||3$. While the intricacies of a taskset could be completely embedded into a more complex optimization program as constraints, because order of operation (i.e. dependency) is a known, unchanging information, it can be utilized to simplify the problem. In the case with the selection of the first task to do, 20 out of the 23 tasks are infeasible tasks to begin with, and even indirectly including them as constraints of other constraints would simply increase the solve time.

2) Kinematic Constraint: Depending on the platform, its unique characteristics could also be leveraged to simplify the optimization. In the case with the dual-arm robot, the most efficient operation may be to always use both arms. However, because of the coupling joint between the two arms as well as the redundancy, there exists kinematic constraints that must be included in the optimization, as depending on the tasks, both arms may not be usable. Since the environment is known, as well as the taskset, this a priori information can again be used to eliminate constraints that are guaranteed to be irrelevant.

Assuming that cooking materials, appliances, and tools are placed at specific locations, it is possible to calculate in advance whether an arm can reach those positions. If the location is within the reachable workspace, the robot can

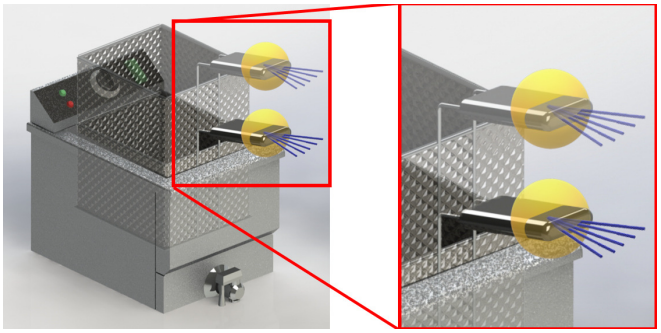

Fig. 3. Deep fryer and reachablilty sphere

reach those places in most cases with one of the two arms. However, when one arm is occupied, it is not guaranteed that the other arm will also be able to reach a desired position that it would have been able to reach had the other arm been unoccupied. This problem which is introduced due to the coupling of the two arms in the body joint acts as a catalyst for pre-computing the entire reachable workspace and building a hashtable like matrix called the Relation Matrix.

One way to calculate the reachable workspace is to discretize the Cartesian space into smaller cubes and verify if all points on a surface of a sphere inside the cube can be reached [15]. However, for cooking tools such as a deep fryer shown in Figure 3, because the point of interest is at the handle of the tool, the amount of volume that needs to be pre-computed for such a tool is in fact very small, reducing the amount of computation required. For example, the yellow spheres in Figure 3 represent the sphere inscribed in the cube representing the handle to conduct the deep fryer task. The blue lines through the sphere are the feasible orientations that the robot can grasp the handle to lift the basket. This example shows 10 possible locations between the two spheres. If the reachability is computed for these points, the feasibility of executing the task can also be known. Such information can be encapsulated in a hashtable-like representation called the Relation Matrix.

Definition A Relation Matrix M_R is a binary matrix where the rows represent one arm's capability to reach $\mathcal{L}_i$ and the columns represent the other arm's capability to reach $\mathcal{L}_i$. If there exists a non-zero element at $R_M(m, n)$, then one arm can reach $\mathcal{L}_m$ while the other arm can also reach $\mathcal{L}_n$.

As an example, Fig. 4 shows a Relation Matrix that can be built for $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$, and $\mathcal{L}_4$. This example assumes that the robot can perform the task as long as the end-effect can reach the rectangular area. In Fig. 4 (a), when the left arm is conducting a task at $\mathcal{L}_1$, the right arm can reach $\mathcal{L}_2$, $\mathcal{L}_3$, and $\mathcal{L}_4$. However in Fig. 4 (b) and (c), the right arm may reach its joint limits or collide with the left arm when reaching for a position such as $\mathcal{L}_2$. This situation and incorporates the kinematic and environmental constraints are mathematically represented in a binary fashion as seen in Fig. 4 (d).

Observing M_R in the running example, when one of

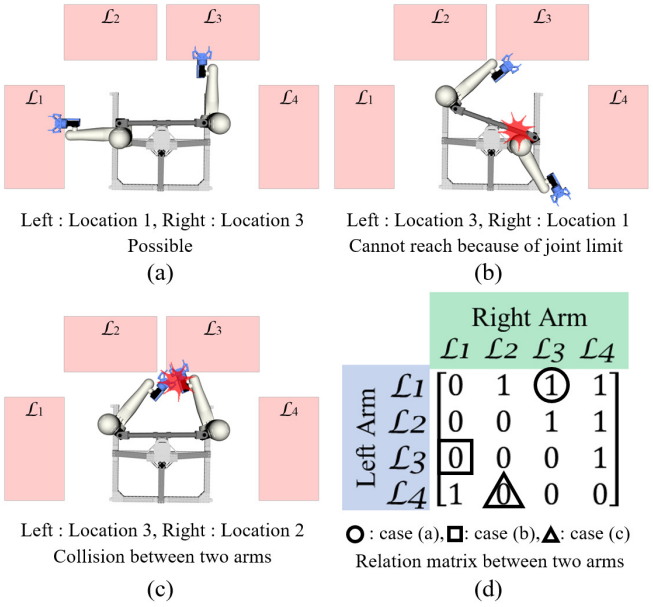

Fig. 4. Examples of tasks that may or may not be performed simultaneously with two arms. (a) Successful case (b) Fail case because of joint limit (c) Fail case because of collision (d) Relation matrix

the arms are occupied, it is clear that the optimization only needs to consider the locations that the other arm can actually reach. Thus, the problem size can again be reduced before the optimization is done as incorporating the full constraints only complicates the problem while not providing any meaningful guards. Note that unlike the pre-processing from the dependency constraints, the Relation Matrix continues to be used in the ensuing framework.

III. Task Scheduling with Mixed-Integer Programming

This section proposes a task scheduling algorithm considering the unique constraints mentioned in section II. The overall algorithm is shown in Algorithm 1. Once a taskset, $\mathbb{T}$, such as the one shown in Table I is prepared, mixed-integer optimization is conducted every timestep to determine the next task for an idle arm until the algorithm exhausts all the task. Because of the aforementioned pre-processing steps taken to reduce the problem size (line 4∼line 10), the optimization can be solved faster, as is explained in Section III-A. Section III-B describes factors for designing an appropriate objective function, and the resulting optimization problem is formulated in Section III-C. After an optimal task is selected, the task's index is obtained (line 22) and the algorithm for the timestep ends.

A. Search Space Reduction

The search space can be cut down by realizing kinematic and dependency constraints discussed in Section II. This process reduces the size of the lists of tasks to search over, v_i and v_f, and eventually the size of the decision vector, $\mathbf{C}$, allowing the optimization to be solved in shorter time.

This process is shown in line 1∼line 10 of Algorithm 1. Given task set,$\mathbb{T}$, contains n tasks, and each task, $\mathcal{T}_i$, has the index of the related dish, $\mathcal{D}_i$, the required time, t_i, the task location, $\mathcal{L}_i$, and the index of prerequisite task, d_i as shown earlier in Table I.

First, OCCUPY() determines which arm is currently in use and queries the location of the arm. M_R is $\mathbb{Z}^{m\times m}$ matrix, where m is the number of given locations. M_R takes the locations of the arm, and determines possible locations the idle arm can take. If both arm are in idle, the same procedure is taken with randomly chosen one. Then, by taking next top-priority tasks across the dishes, the dependency constraint considered. Lastly, CHOICE$(\mathcal{L}, LR)$ calculates the number of possible locations that the other arm would have in the next time. This number works as the degree of flexibility as described in item 2 of Section III-B.

B. Factors for the Objective Function

The following two factors are mainly considered to design the objective function.

1) Required time for each task: If only one arm takes excessive amount of time and it is at a location that forces the other arm to stay idle, this will cause extra time to finish the entire list of dishes. Therefore, executing tasks that require shorter time instances a preferable to increase the odds of both arms concurrently working.
2) The degree of flexibility on task selection: When the scheduler decide which task is an optimal for an idle arm, it is preferable to consider the other busy arm's flexibility in selecting the next task. This would promote concurrent execution of both arms in future. In our formulation, the number of available locations of tasks in the next optimization step for the other busy arm is used as a measure of the degree of flexibility. The measure is scored by CHOICES$(\mathcal{L}, LR)$ in Algorithm 2 and summing up the possible number of locations of M_R, which implies that the dependency and kinematic constraints are also considered.

C. Mixed-Integer Formulation and Task Selection

To solve this assignment problem, a mixed-integer optimization is formulated as shown in Equation (1).

$$\begin{aligned}
\arg\max_{\mathbf{c}} \quad & \alpha\mathbf{c}^T\mathbf{v}_t + (1-\alpha)\mathbf{c}^T\mathbf{v}_f && \text{(1a)}\\
\text{subject to} \quad & \mathbf{c} \in \mathbb{Z}^k, && \text{(1b)}\\
& \mathbf{0}_{k\times 1} \le \mathbf{c} \le \mathbf{1}_{k\times 1}, && \text{(1c)}\\
& \sum \mathbf{c} = 1 && \text{(1d)}
\end{aligned}$$

1) Objective Function: The objective function comprises two weighted terms. $\mathbf{v}_t$ contains reciprocal of time required of selected tasks and is a realization of Item 1. $\mathbf{v}_f$ is a vector with the number of locations of tasks that the other arm can take in the next optimization when corresponding task is selected for this arm, and is a realization of Item 2. $\mathbf{c} \in \mathbb{Z}^k$ is a the decision vector, and k is the number of potential tasks

already filtered as in Section III-A. α is the relative weight between the two objective terms. Section IV describes how these weights affect the results.

2) Constraint: In this assignment problem, 1 is assigned when the corresponding task is selected as the next optimal task, 0 otherwise. Equation (1b) and Equation (1c) constrain each element of decision vector, $\mathbf{c}_{i,1}$, to be either 1 or 0. Equation (1d) represents the constraint that there is only one task to be selected as an optimal task.

Algorithm 1 Task Planning with MIP

$\mathbb{T} = \{\mathcal{T}_i\} = \{(\mathcal{D}_i, t_i, \mathcal{L}_i, d_i)\}$
$n \leftarrow$ size of $\mathbb{T}$;
while $\sum_{i=1}^{n} t_i > 0$ **do**
  $lr \leftarrow$ OCCUPY()
  for i=1 to n **do**
    if $lr = L$ or $lr = R$ **then**
      $TF \leftarrow (lr = L)?M_R(\mathcal{L}_L, \mathcal{L}_i) : M_R(\mathcal{L}_i, \mathcal{L}_R)$
      if d_i has top priority and $TF = true$ **then**
        $\mathbf{v}_i.append(i)$; $\mathbf{v}_t.append(1/t_i)$;
        $\mathbf{v}_f.append(\text{CHOICES}(\mathcal{L}_i, lr))$
    else if $lr = \phi$ **then**
      if d_i has top priority **then**
        $\mathbf{v}_i.append(i)$; $\mathbf{v}_t.append(1/t_i)$;
        if $\text{CHOICES}(\mathcal{L}_i, L)) \neq 0$ **then**
          $\mathbf{v}_f.append(\text{CHOICES}(\mathcal{L}_i, L))$
        else
          $\mathbf{v}_f.append(\text{CHOICES}(\mathcal{L}_i, R))$
    else
      pass
  find $\arg\max_{\mathbf{c}\in\mathbb{Z}^n} \alpha\mathbf{c}^T\mathbf{v}_t + (1-\alpha)\mathbf{c}^T\mathbf{v}_f$
    subject to $\sum \mathbf{C} = 1$
  $c \leftarrow \mathbf{c}^T\mathbf{v}_i$
  $t_c \leftarrow t_c - ts$

Algorithm 2 Choice Options for Arm

procedure CHOICES($\mathcal{L}, lr$)
  $m \leftarrow size(M_R)$
  if $lr = L$ **then**
    $sc \leftarrow \sum_{i=1}^{m} M_R(\mathcal{L}, i)$
  else if $lr = R$ **then**
    $sc \leftarrow \sum_{i=1}^{m} M_R(i, \mathcal{L})$
  return sc

IV. Validation Test

In this section, the results from testing Algorithm 1 under a series of test conditions are presented. These conditions are generated in the following order:

1) With number of $\mathcal{T}$ as constant, t and $\mathcal{L}$ are randomized following a normal distribution.
2) The size of M_R is modified according to the number of $\mathcal{L}$.
3) The elements of M_R are randomized to 0 or 1.

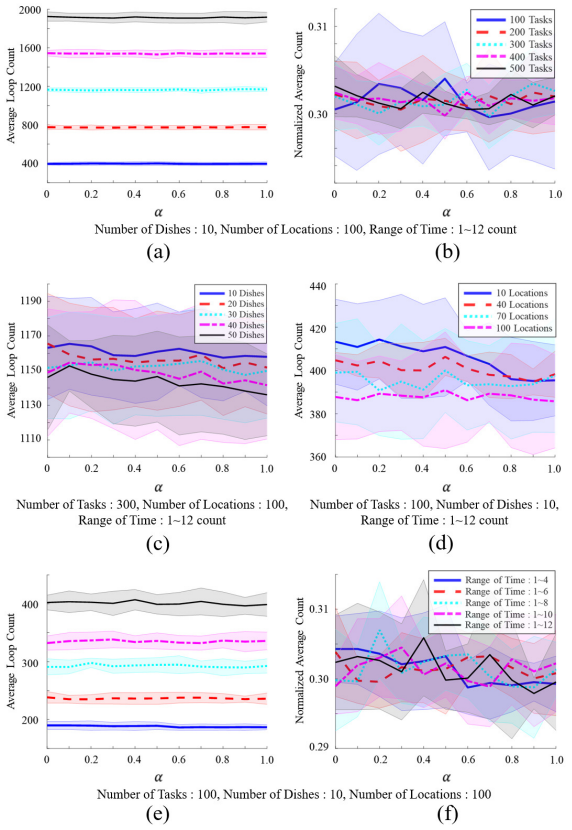

Fig. 5. Validation test results. (a) Result when the number of tasks varies. (b) Normalized result when the number of tasks varies. (c) Result when the number of dishes varies. (d) Result when range of the number of location varies. (e) Result when range of the number of time required varies. (f) Normalized result when range of the number of time required varies.

4) Time to completion is determined by count, a number indicative of how many times the loop was performed. Thus, ts in Algorithm 1 becomes 1.
5) Number of $\mathcal{D}$ is decided and $\mathcal{T}$ is divided by the number of $\mathcal{D}$.
6) Number of $\mathcal{T}$ per $\mathcal{D}$ is also randomly set.
7) Dependency between $\mathcal{T}$ within a $\mathcal{D}$ is defined.

According to the above method, since the dependency is naturally determined when the dish is decided, the result is observed by changing the remaining four factors: the number of tasks, dishes, range of locations, and time. After the taskset is created, the results are observed by varying the two factors from Section III-C.1: time α and flexibility β factors. The sum of α and β is 1, and α is incremented from 0 to 1 in 0.1 steps. The optimization solver used is MOSEK [16], running on a desktop computer with a CPU of Intel Core i7 8700K @ 3.70GHz and 16GB of RAM at 1204.2MHz.

A. Variation of the number of tasks

The results when changing the number of tasks while all other conditions are kept constant are presented in Fig. 5 (a) and (b). Data were obtained from 20 tasksets, but with variations in the number of tasks. The average and the

TABLE II

AVERAGE RUN-TIME OF EACH EXPERIMENTS.

Number of Task	100	200	300	400	500
Run-Time (s)	136.6	279.2	413.5	546.1	683.4
Number of Dishes	10	20	30	40	50
Run-Time (s)	669.7	713.4	766.2	831.8	905.5
Number of Locations	10	40	70	100	
Run-Time (s)	153.9	143.9	142.7	153.3	
Range of Time (cnt)	1∼4	1∼6	1∼8	1∼10	1∼12
Run-Time (s)	133.3	135.4	132.4	138.6	144.3

standard deviation are shown. In (a), as the number of tasks increases, the total time taken increases, which is an expected result. To seek a potential trend, the results were normalized as seen in (b). In all results, there was no particular trend for the change in α value.

B. Variation of the number of dish

The result when changing the number of dishes while all other conditions were held constant are shown in Fig. 5 (c). 20 tasksets are used, but with variations in the number of dishes. The overall trend is that the larger the value of α, the smaller the total time was consumed. An interesting result was that as the number of dishes increases, the number of loop counts decreases. The reason for this is that as the number of dishes increases, the number of options that the robot arm can choose increases, resulting in the rate at which both arms can be used simultaneously.

C. Variation of the number of location

When creating a taskset, the location is randomly assigned. In this case, the location is given randomly in a different range and the results are compared. As the range increases, the overall loop count tends to decrease. That is because size of the relation matrix become also bigger. The larger the size of the matrix, the higher the probability that the result of the function in algorithm 2 will be larger. And unlike other results, the larger the range, the greater the effect of the flexibility factor. In Fig. 5 (d), the slope of the graph decreases as the location increases is shown. This is because the robots have more choices as the locations become more vary. 20 tasksets are used at each variation of the number of location in simulation.

D. Variation of the range of the time required

The results when changing the range of the time required is in Fig. 5 (e) and (f). The overall trend is similar to that of section IV-A. As the range of time increases, the total time taken(Loop counts) increases. 20 tasksets are used at each variation of the range of the time in simulation.

Putting together all the results in the section IV-A to IV-D, the smaller the total number of tasks, the shorter the time required for each task, and the greater the number of dishes and locations, the probability of obtaining optimal results increases. The average run-time for each case is shown in the Table II.

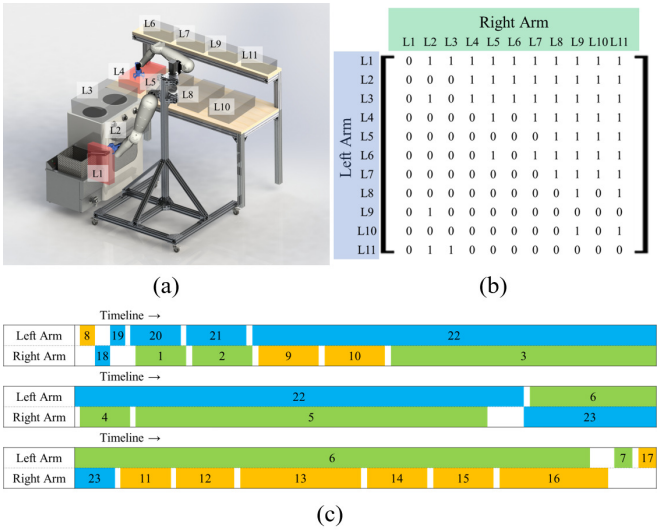

Fig. 6. Result of validation test in real-sized testset. (a) The locations in the workspace. (b) Relation matrix of robot configuration. (c) Selection of tasks in timeline.

E. Validation for real-sized taskset

The previous results have a large number of tasks, making it difficult to see whether our algorithm makes the optimal choice. Therefore, the validation test was conducted for real-sized taskset to make it easier to see the results. For the taskset, Table I is used as input of test. Fig. 6 (a) shows the locations in the workspace like kitchen. It has 11 pre-defined locations, and inverse kinematics can be solved for each location. The relation matrix in (b) was got through this kinematics solutions. The result of test is shown in (c), and it shows the task performed by the left and right arms along the timeline. Three dishes were colored differently. The results show that the sequence of all tasks satisfy the dependency and kinematics constraints, and maximum use of both arms simultaneously.

V. CONCLUSIONS

In this paper, we proposed a framework that can optimally schedule multiple cooking tasks with concurrency for a dual armed robot. A mixed-integer program is formulated to assign an optimal next task which the robot can refer to and execute. A task sequence that minimize cooking time and maximize the number of options an arm to choose in the next optimization step is considered more optimal. The number of decision variables is reduced by taking only feasible tasks by considering the kinematic constraints and sequential constraints unique to cooking.

The approach was validated on a range of test cases to verify the feasibility and scalability of the approach. While this work specifically targeted task scheduling for a cooking application, it can also be used in any situation where more than one robot is sharing the same workspace and coordination between them needs to be optimized for.

Future work includes incorporating constraints that represent tasks that require both arms to be used for a single

task (i.e. bimanipulation, robot cooperation) and optimization over a limited time horizon, rather than only over a single timestep, which would naturally make the approach more optimal and reactive to unpredictable failure in task execution.

REFERENCES

[1] J. L. Jones, "Robots at the tipping point: the road to irobot roomba," *IEEE Robotics & Automation Magazine*, vol. 13, no. 1, pp. 76–78, 2006.

[2] M. V. Minniti, F. Farshidian, R. Grandia, and M. Hutter, "Whole-body mpc for a dynamically stable mobile manipulator," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3687–3694, 2019.

[3] "Spot®." [Online]. Available: https://www.bostondynamics.com/spot

[4] C. D. Bellicoso, K. Krämer, M. Stäuble, D. Sako, F. Jenelten, M. Bjelonic, and M. Hutter, "Alma-articulated locomotion and manipulation for a torque-controllable robot," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8477–8483.

[5] M. Caraher and T. Lang, "Can't cook, won't cook: A review of cooking skills and their relevance to health promotion," *International Journal of Health Promotion and Education*, vol. 37, no. 3, pp. 89–100, 1999.

[6] "The future of food is here." [Online]. Available: https://misorobotics.com/

[7] K. Wiggers, "Miso robotics unveils its next-gen robot kitchen assistant," Jan 2020. [Online]. Available: https://venturebeat.com/2020/01/28/miso-robotics-unveils-its-next-gen-robot-kitchen-assistant/

[8] "Samsung bot chef at ces 2020." [Online]. Available: https://www.youtube.com/watch?v=OwA6-b1Z7aQ

[9] "(eng) 'cloi's table zone', futuristic restaurant at ces 2020." [Online]. Available: https://www.youtube.com/watch?v=vsZ_HUAPXL8

[10] F. I. Craik and E. Bialystok, "Planning and task management in older adults: Cooking breakfast," *Memory & Cognition*, vol. 34, no. 6, pp. 1236–1249, 2006.

[11] A. J. Garden, "Systems and methods of preparing food products," USA Patent US9 292 889B2, 2016.

[12] K. E. Booth, T. T. Tran, G. Nejat, and J. C. Beck, "Mixed-integer and constraint programming techniques for mobile robot task planning," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 500–507, 2016.

[13] 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.

[14] S. Kim, K. Jang, S. Park, Y. Lee, S. Y. Lee, and J. Park, "Continuous task transition approach for robot controller based on hierarchical quadratic programming," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1603–1610, 2019.

[15] F. Zacharias, C. Borst, and G. Hirzinger, "Capturing robot workspace structure: representing robot capabilities," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Ieee, 2007, pp. 3229–3236.

[16] E. D. Andersen and K. D. Andersen, "The mosek interior point optimizer for linear programming: an implementation of the homogeneous algorithm," in *High performance optimization*. Springer, 2000, pp. 197–232.