REAL-TIME SCHEDULING OF DISTRIBUTED MULTI-ROBOT MANIPULATOR SYSTEMS

P. Yuan, M. Moallem and R. V. Patel
Department of Electrical and Computer Engineering, University of Western Ontario,
London, Ontario, Canada
Contact: pyuan@ engga.uwo.ca

Received September 2004, Accepted February 2005
No. 04-CSME-53, E.I.C. Accession 2843

Abstract

This paper presents an online task-oriented scheduling method and an off-line scheduling algorithm that can be used for cooperative control of a distributed multi-robot manipulator system. Satisfaction of temporal deadlines and tasks-relative constraints are considered in this work. With the proposed algorithms, both the timing constraints and relative task dependencies can be satisfied when the worst-case execution time is unknown. The total execution time of the assembly tasks can be significantly improved compared with other known scheduling algorithms such as the First-In-First-Out and Round Robin scheduling methods. Experimental results are presented indicating that the proposed algorithm can be used for improving the performance of multi-robot systems in terms of timing and resource constraints.

LES PLANIFICATIONS EN TEMPS REEL DE SYSTEMES DE MANIPULATEUR DE MULTI-ROBOT DISTRIBUES

Résumé

Ce papier présente une méthode de planification en ligne adapté à la tâche et un algorithme de planification autonome qui peut être utilisé pour le contrôle coopératif d'un système de manipulateur de multi-robot distribué. La satisfaction de temps limite et les contraintes relatives aux tâches sont considérées dans ce travail. Avec les algorithmes proposés, les contraintes de temps et les dépendances aux tâches relatives peuvent être satisfaites quand le temps d'exécution des cas les pires est inconnu.Le temps total d'exécution des tâches d'assemblée peut être amélioré significativement comparé aux autres algorithmes de planification connus tels que le « First-In-First-Out » et les méthodes de planification de « Round Robin ». Les résultats expérimentaux sont présentés indiquant que l'algorithme proposé peut être utilisé pour améliorer l'exécution de systèmes de multi-robot sur le plan des contraintes de temps et de ressources.

1 INTRODUCTION

Most multi-robot systems are hard real-time systems requiring the use of real-time operating system and hence a real-time scheduler. The real-time scheduler depends significantly on the underlying dispatching mechanism. There are two dominant dispatching mechanisms: time-driven and priority-driven. In time-driven dispatching, each task has a start time associated with it. For priority driven algorithms, dispatching depends on the priority of a task. Priority assignment can be static or dynamic [2]. Schedulers may be preemptive or non-preemptive and execute tasks according to priority assignments.

Many scheduling algorithms have been published in the literature. Branch and bound search is commonly used for optimal scheduling of time-driven scheduling systems [5]. Other schemes use priority-driven scheduling, such as First-In-First-Out (FIFO), Round-Robin (RR), Earliest-Deadline-First (EDF), Minimum-Laxity-First (MLF), Least-Slack-Time-First (LST), etc [6]. The task latency is considered in [3] and a multiprocessor fixed-priority scheduling is presented in [18]. The scheduling algorithms are widely used and can satisfy most scheduling requirements [7]-[14].

Unfortunately, none of the above schemes considered task dependencies which are important in multi-robot assembly systems. Task dependencies (also known as relative constraints) are considered in [6] and a parametric scheduling method has been developed for the special case of multi-level chains of tasks. Similar results have been obtained for tasks with relative separation constraints [11]. Design-to-time is an AI approach for solving real-time problems when multiple methods that make tradeoffs in execution time and solution quality are available for many tasks [7]. Design-to-time involves designing a solution to a problem that uses all available resources to maximize the solution quality within the available time. However, the method does not consider task dependencies and therefore may not be applicable to a multi-robot system. For example, in a multi-robot assembly system with many parts, if we define the assembly of each part as a task, the task sequence will be the key issue for scheduler. In our previous work [19], task dependency is considered and a task-oriented algorithm is addressed. In this paper, deadlock situation is considered and an offline scheduling algorithm is proposed to avoid deadlock situations. The schedulability analysis of this method is aslo discussed.

This paper is organized as follows. Section 2 presents the motivation for our scheduling algorithms. Section 3 discusses the task interdependency and presents the Task-Oriented Algorithm (TOA). An offline scheduling algorithm that can solve the dead-lock situations is discussed in Section 4. An example to illustrate the algorithm together with experimental results are given in Section 5. Conclusions and future work are given in Section 6.

2 PROBLEM STATEMENT

The problem we consider here is to find the execution order of n tasks with m timing constraints for p robot systems.

2.1 Scheduling Problems

Here we define a real-time task system Λ with p robots and n tasks, $(n \geq p \geq 1)$, as a 5-tuple $\{R, C, S, F, P\}$ as follows:

- $R = \{R_1, \cdots, R_n\}$ is the set of dispatch times for the n tasks.
- $C = \{C_{j,1}, \cdots, C_{j,n}\}$ is the set of execution time, where $C_{j,i}$ is the execution time of task i on robot j $(j = 1, \cdots, p)$. If all robot tasks are identical, then $C = \{C_1, \cdots, C_n\}$, where C_i is the

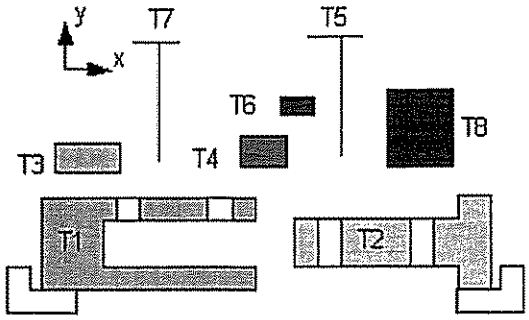

Figure 1: Assembly sequential and fixture requirements

execution time of task i on any robot. The execution time is unknown in advance and may vary due to network communication delays or other issues.

- $S = \{S_1, \cdots, S_n\}$ is the set of starting times for the n tasks. Each task starts executing no later than the corresponding times in the set.
- $F = \{F_1, \cdots, F_n\}$ is the expected finish time for the n tasks. Each task has to be finished before these times which are the hard deadlines for the n tasks.
- $P = \{P_1, \cdots, P_n\}$ is the periods for the n tasks, with $0 < C_i < F_i = P_i$. The set P is relevant only for periodic real-time systems.

In a real-time system consisting of multiple tasks, the 5-tuple $\{R_i, C_i, S_i, F_i, P_i\}$ specifies each task. Such a system can have periodic or non-periodic tasks. Each task can have a distinct priority or share the same priority with other tasks. The start time and finish time of each task are defined in advance which means that the task must be triggered before the start time and be finished before the deadline despite varying execution times. All timing constraints and the relative task dependencies must be satisfied.

2.2 Scheduling Constraints Classification

Two constraints that apply to all the scheduling problems are considered here: Task constraint and robot constraint. The task constraint requires that, at any time, no task can be executed by more than one robot. The robot constraint requires that, at any time, no robot can work on more than one task. Other constraints that uniquely distinguish a particular scheduling problem are as follows:

- If task k must be done immediately before task p, task k is called the father of task p.
- If task k must be done immediately after task p, task k is called the son of task p.
- If task k has no relationship with task p, we call that task k is irrelevant to task p.
- A task is triggered only if all its father tasks have been done.
- All tasks are non-preemptive. The reason for this constraint will be discussed later.

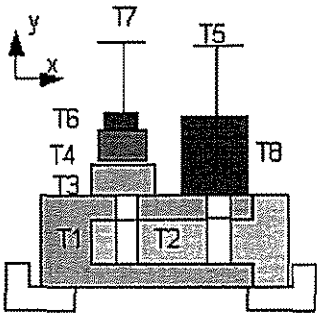

Figure 2: Expected assembly product

2.3 Example Of Scheduling Problems

To illustrate a typical scheduling, let us study a simple assembly task. Consider an automatic assembly system with eight parts as depicted in Fig. 1 and Fig. 2. The assembly process can be divided into the following unit operations:

- T1: Take part 1 to a fixture in the x direction.
- T2: Insert part 2 in the -x direction.
- T3: Insert part 3 in the y direction.
- T4: Insert part 4 in the y direction.
- T5: Insert part 5 in the y direction.
- T6: Insert part 6 in the y direction.
- T7: Insert part 7 in the y direction.
- T8: Insert part 8 in the y direction.

Each task has a set of operations, such as move the robot arm 1 to the calculated Cartesian position p1; open the gripper and get part 1; close the gripper; move to the fixture position f1, etc. Fig. 1 shows the assembly components of the process with the expected assembly result shown in Fig. 2. Eight parts need to be assembled as fast and accurately as possible. For a single robot system, this process is simple since there is no redundancy. The robot will pick up all parts one by one and assemble them in order. Even for a multi-robot system, if the execution time of each part is known, any offline scheduling algorithm can be used to create the exact execution sequence. However, considering the communication time and network latency, each robot with a different speed may finish its task at a different time. Therefore, the scheduling algorithm should be dynamic and online instead of offline. Furthermore, there are some implicit constraints and relationships in this simple example. For example, there could be two parallel execution streams with a locus at Task T3. Assume that T3 must be done as soon as possible after both T1 and T2 are transferred to the fixture. Then T3 will have a higher priority than T8 despite being listed as a parallel step.

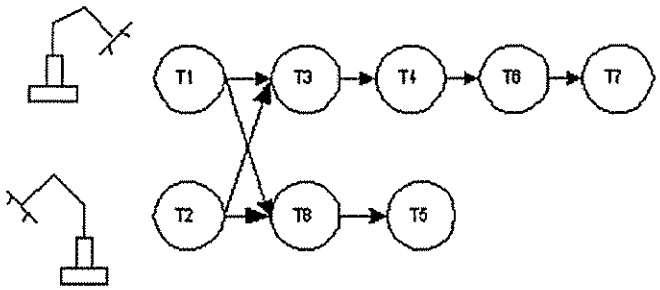

Figure 3: Tasks scheduling requirements and execution orders

3 THE TASK ORIENTED ALGORITHM (TOA)

We are interested in obtaining the required dispatch time of the eight tasks for the problem statement given in Section 2. The scheduler determines which task should be run first when there is a potential conflict. For example, consider the timing constraints of this system for the case where there are two parallel execution steps as follows:

- Robot 1: $T1 \rightarrow T3 \rightarrow T4 \rightarrow T6 \rightarrow T7$
- Robot 2: $T2 \rightarrow T8 \rightarrow T5$

As we do not know how much time is required for each robot to finish its current task, this kind of division may not be efficient. Given the timing constraints, it can be concluded that the scheduling methods such as EDF and/or FIFO do not consider task dependencies. A task-oriented-scheduler (TOA) is discussed here to resolve this problem which provides the execution order for the tasks. A possible execution sequence is shown in Fig. 3.

3.1 Task Model

As shown in Fig. 3, the scheduling requirements cannot be easily satisfied without considering the task interdependency due to uncertain execution times. So we need to consider both the timing requirements and task interdependency to keep the exact execution order.

Therefore, we give the task model here. Consider a system consisting of n tasks $\Lambda = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task is defined by a set $\tau_i = \{e_i, s_i, f_i, p_i\}$, where:

- e_i: is the period if the task is periodic; otherwise e_i=0;
- s_i: start time of each task;
- f_i: finish time of τ_i
- p_i: the priority, if there is any

3.2 Timing Constraints

The task model allows for defining different kinds of timing constraints. The above sets define a task model to describe timing constraints of individual tasks. A realistic timing constraint can be modeled

between two tasks as follows:

$$\begin{pmatrix} s_i \leq s_j + c_1 & (1) \\ s_i \leq f_j + c_2 & (2) \\ f_i \leq s_j + c_3 & (3) \\ f_i \leq f_j + c_4 & (4) \\ s_i \leq c_5 & (5) \\ f_i \leq c_6 & (6) \end{pmatrix}$$

where (1) means task i cannot start till after task j starts; (2) means task i cannot start until after task j finishes; (3) means task i cannot finish until after task j starts; (4) means task i cannot finish until after task j finishes; (5) means task i cannot start earlier than time $t = c_5$; and (6) means task i cannot finish after time $t = c_6$.

Note that here we only consider the relationships between two tasks. Those relationships are related to the start time s_i and the finish time f_i.

Let the vector χ contain the start and finish times of the tasks, i.e,

$$\chi = \begin{bmatrix} s_1 \\ f_1 \\ \vdots \\ s_n \\ f_n \end{bmatrix}$$

and A be defined as:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,2n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,2n} \\ & & \ddots & \\ a_{m,1} & a_{m,2} & \cdots & a_{m,2n} \end{bmatrix}$$

where:

$$a_{i,j} = \begin{cases} 1 & \text{if task i is released before task j;} \\ 0 & \text{if task i and task j have no dependency;} \\ -1 & \text{if task i is released after task j.} \end{cases}$$

and let C represent a timing constraint given by:

$$C = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

. Then, for n tasks with m timing constraints, using the timing constraints equation, we have:

$$A \cdot \chi \leq C$$

where A is an $m \times 2n$ matrix, χ is a $2n$-dimensional column vector and C is an m-dimensional column vector of real-valued constraints.

```
Begin
  ULONG task_reg = 0x0; //initialize task register
  Initialize_tasks();
  Do i =1 : n
      For (each task Ti ) Do
           Get_Shared_Memory(&Task_reg);
           If ((si < ui) & (status == exec))
              Execute(Ti);
              Task_reg = task_reg | i;
              Update(execution_order);
           Otherwise continue;
           Add(new task_dependencies);
       While (task_reg & reg == reg);
    //if meets all dependencies
  End
End
```

Figure 4: Implementation of the task-oriented-algorithm (TOA)

3.3 The Task-Oriented-Algorithm

Task interdependency determines the required execution order despite the timing deadline, task priority, etc. This interdependency is always positive, which means that each task expects others to follow without preemption. For example, as mentioned above, T3 can only be executed when T1 and T2 are both done. So we say T3 relies on T1 and T2. The schedulability is defined as follows:

Definition: Given a set of tasks $\Lambda = \{\tau_1, \tau_2, \ldots, \tau_n\}$, where $\tau_i = \{e_i, s_i, f_i, p_i\}$, this task set is schedulable if there exists a set $S = (t_{i_1}, t_{i_2}, \ldots, t_{i_n})$ which satisfies both the timing constraints and task dependencies.

The definition of scheduability shows that the task sets can be scheduled only if both the deadline $\{e_{i_j} \in (S_{i_j}, f_{i_j}), 1 \leq j \leq n-1\}$ and the task dependencies $\{S_{i_j} + u_{i_j} \leq S_{i_{j+1}}, 1 \leq j \leq n-1\}$ are satisfied together. Fig. 4 gives an implementation of the task-oriented algorithm. Referring to Fig. 4, each task is assigned a vector at the initialization time $T_i = \{N_i, d_i, p_i\}$, where p_i, d_i and N_i represent the priority, task dependencies and execution order respectively. Each task has three states: 'ready', 'waiting' and 'execution'. A global register represents the current task's execution state. If any task has been done, the register will be updated so that other tasks will know that. Therefore, the dependent tasks will be switched to 'ready' status.

Each task reads the shared memory periodically to see whether the father task/tasks have been completed before it is dispatched and updates the shared memory once it is finished. Since it costs much more time for the robot to finish a task than it takes for the CPU to periodically check the register, we assume that the checking time of each task can be neglected. In this case, the required scheduling sequence and system safety can both be satisfied by using the global and local variables if there exists an execution order. Robots talk to each other using shared memory to avoid deadlock and priority reversion.

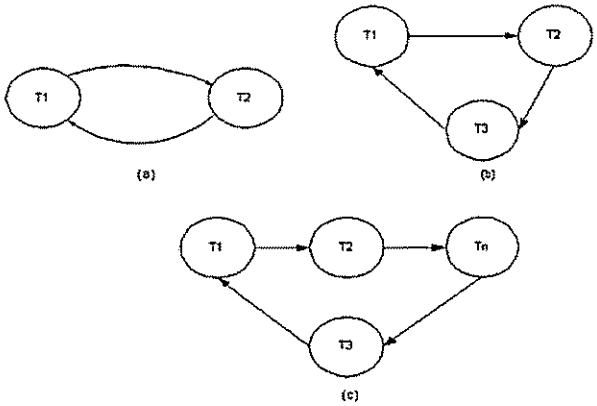

Figure 5: Possible deadlock between/among tasks: (a) two tasks (b) three tasks (c) more tasks

4 OFFLINE SCHEDULING ALGORITHM

4.1 Deadlock

The deadlock situation can arise, for example, when two tasks have been blocked and depend on each other to continue their operations. This could also be true in resource competition. Deadlock can also appear in three tasks or more tasks. Fig. 5 illustrates the possible deadlock situations.

4.2 Possible Task Dependencies between Two Tasks

As defined above, there are three possible relationships between two tasks: task i is the father of task j; task i is the son of task j; no relationship between task i and task j. Therefore, the dependency expression of the task dependencies are as follows. If:

$$f1 < s2 \qquad A = \begin{bmatrix} 0 & 1 & -1 & 0 \end{bmatrix}$$

which means task 2 starts after task 1 finishes. Obviously we know that task 2 starts after task 1 starts. So we have:

$$s1 < f2 \qquad A = \begin{bmatrix} 1 & 0 & 0 & -1 \end{bmatrix}$$

However, possible deadlock situations happen when there are the following mistakes:

$$\begin{matrix} f1 < s2 \\ f2 < s1 \end{matrix} \qquad A = \begin{bmatrix} 0 & 1 & -1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}$$

This is called a deadlock. The system will fail to execute the tasks since both task 1 and task 2 need to start first. Therefore, deadlocks must be avoided.

Lemma 4.1 *Given a system:*

$$A \cdot \chi \leq C$$

If $\forall i, j,$

$$a_{i,2k}a_{j,2p} = a_{i,2k-1}a_{j,2p} = a_{i,2k}a_{j,2p-1} = a_{i,2k-1}a_{j,2p-1} = c$$
$$(i, j, k, p = 1, 2, \cdots, n, k \neq p)$$

is satisfied, then the task dependency between two tasks is satisfied.

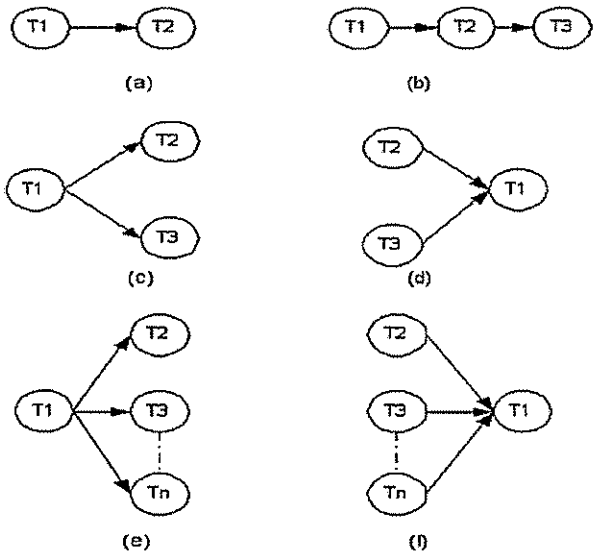

Figure 6: (a) Task dependency (b) sequential task dependency (c) cross interdependency (d) extended interdependency (e) ex-cross interdependency (f) ex-extended interdependency

Proof 4.2 : *1) If there is task dependency between task* m, *and task* p, *from the given requirements, we know that: If* $k < p$, *from the task dependency definition, we can always get:*

$$s_k < s_p, \qquad f_k < f_p, \qquad s_k < f_p, \qquad f_k < s_p$$

Obviously, from the definition of the expression matrix A, we get:

$$a_{i,2k}a_{j,2p} = a_{i,2k-1}a_{j,2p} = a_{i,2k}a_{j,2p-1} = a_{i,2k-1}a_{j,2p-1} = 1$$

otherwise, we have:

$$a_{i,2k}a_{j,2p} = a_{i,2k-1}a_{j,2p} = a_{i,2k}a_{j,2p-1} = a_{i,2k-1}a_{j,2p-1} = -1$$

2) If there is no task dependency between task m and task p, then:

$$a_{i,2k}a_{j,2p} = a_{i,2k-1}a_{j,2p} = a_{i,2k}a_{j,2p-1} = a_{i,2k-1}a_{j,2p-1} = 0$$

This method can be extended to deadlock situations among three or more tasks as illustrated in Fig. 6. Furthermore, it is easy to be used in computer programming. Therefore, the proposed scheduling algorithm is suitable for large number of tasks with more possible deadlock situations.

5 EXPERIMENTAL RESULTS

To test the algorithms, we set up a series of experiments. Our system uses Windows 2000, Windows NT, Tornado, and the VxWork real-time operating systems running on Pentium 3 computers. The robots used for these experiments were a seven degree of freedom CRS-F3 robot and a six degree of freedom CRS-F3 robot. Each arm is connected to a controller and a force/torque sensor. Each arm has a 65 mm servo gripper that is fastened to the tool flange. Average Cycle Time of the gripper (from closed to open)

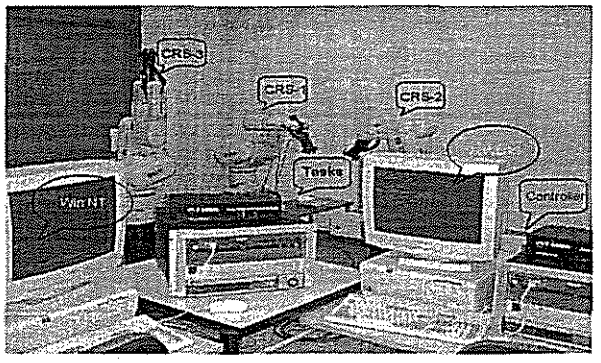

Figure 7: Overview of distributed multi-robot system test bed

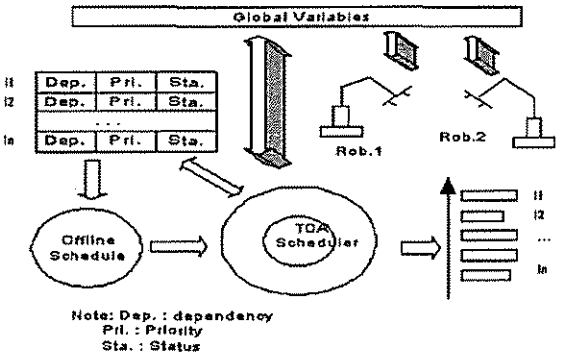

Figure 8: Overview of the TOA and communication in two robots system

is 1.23 seconds. Therefore, for each robot task with actions like grasp and release, the time required must be longer than this period. The maximum holding force is 3.75 KgF (1 KgF = 9.8 N). Fig. 7 shows the layout of the test bed. Shown in the picture are CRS-F3 robot arms, controller, the force/torque sensor, and environment used in the experiments. The workspace was a board with several parts to be assembled. Here we define the unit actions as: MoveToPoint(), open_grip(), close_grip(), calrdyRobot(), e_stop(), sendRobotCmd(), approPoint(), departPoint(), set_speed, open_port(), close_port(), etc. Each task is composed of one or more unit actions.

5.1 Scheduling with First-In-First-Out (FIFO)

Two scheduling algorithms are widely used in multitasking operating systems: First-In-First-Out (FIFO)(also known as First-Come-First-Serve (FCFS)), and Round-Robin (RR). FIFO is a relatively simple but widely used algorithm. As discussed before, two main issues must be considered here. First, the execution time of each task is not known *a priori* and may be vary; secondly, the task dependencies must be satisfied. In other words, no task can be loaded in until all its timing constraints and relative constraints are satisfied. Therefore, to schedule with FIFO, the worst case execution time of each task must be known or estimated. Table 1 gives the experimental average execution time of the eight tasks. From Table 1, the worst-case execution time of the eight tasks are estimated as Table 3. Each task was spawned at exactly its dispatch time. Table 2 gives the experimental results of the execution time and

No.	1	2	3	4	5	6	7	8	aver	std
T1	767	628	794	622	762	743	670	738	716	66
T2	2536	2406	2453	2583	2403	2579	3006	3125	2636	276
T3	3154	3288	3202	3227	3134	3122	3159	3216	3188	56
T4	2431	2485	2451	2457	2553	2433	2448	2414	2459	43
T5	2505	2403	2496	2494	2518	2401	2590	2534	2493	64
T6	2545	2543	2467	2545	2520	2536	2557	2462	2522	37
T7	382	467	454	379	344	460	329	416	404	54
T8	2524	2565	2561	2551	2562	2471	2406	2581	2528	60

Table 1: Average execution time of the tasks (unit: 1 tick = 1/60 second).

No.	T1	T2	T3	T4	T5	T6	T7	T8	Total
aver	697	2532	3186	2524	2512	2467	2495	388	9688
std	67	51	44	60	64	62	60	57	57

Table 2: Average execution time of the tasks with FIFO (unit: 1 tick = 1/60 second).

the start time of each task. Note that here we assume that the worst case execution times are as shown in Fig. 9. If the worst-case condition is different in this list, the FIFO may fail in scheduling the task sets.

5.2 Scheduling with Round-Robin(RR)

Round-Robin (RR) is similar to FIFO, but lets the tasks share the CPU in time slices. Table 4 gives the experimental results for the average execution time and standard derivation for each task. Compared with Table 2, we can see that, both the timing constraints and task dependencies are satisfied.

5.3 Scheduling with the Task-Oriented-Algorithm (TOA)

As discussed before, we assume that the worst case execution time of each task is known or can be estimated when FIFO or RR scheduling is used. The situation, however, is not easily satisfied since we cannot consider all the worst cases. In what follows will discuss the scheduling TOA algorithm where

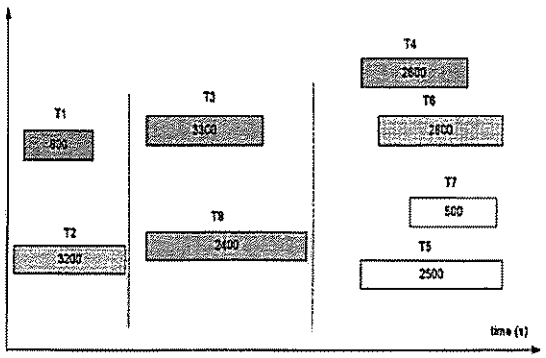

Figure 9: Worst case execution time of the eight tasks

t = 0	T1 and T2
t = 800	T3
t = 3200	T8
t = 4100	T4
t = 5800	T5
t = 6700	T6
t = 9300	T7

Table 3: Dispatch time of the tasks (unit: 1 tick = 1/60 second).

No.	T1	T2	T3	T4	T5	T6	T7	T8	Total
aver	727	2504	3187	2509	2508	2517	2507	395	9695
std	26	65	59	57	38	65	64	74	73

Table 4: Average execution time of the tasks with RR (unit: 1 tick = 1/60 second).

knowledge of the worst case execution time of each task is not required.

5.3.1 Step 1: Initialization of Tasks

In the initialization step, eight unit tasks identified as $T_1, T_2, T_3, \ldots, T_8$ are spawned at first. Each task performs the part assembly discussed in Section 2 and is initialized with timing constraints and relative task dependencies. To test the algorithm, all the tasks are dispatched at t=0, which means that the scheduler will load the tasks in order to meet all the deadlines and provide the correct sequential order as shown in Fig. 3. From the given requirements, we can get the task model as follows:

- UINT task_reg=0; /* 0000 0000 0000 0000 0000 0000 0000 0000*/
- T1={0x0,1,0} /* dependency=0, priority=1, status=0 */
- T2={0x0,1,0}
- T3={0x3,1,0} /* depends on T1 and T2(011) */
- T4={0x4,1,0}/* depends on T3 (100) */
- T6={0x8,2,0} /*depends on T3 (100) */
- T7={0x20,1,0} /*depends on T6 (10 0000)*/
- T8={0x3,1,0} /* depends on T1 and T2 (011) */
- T5={0x80,1,0} /*depends on T8 (1000 0000)*/

No.	T1	T2	T3	T4	T5	T6	T7	T8	Total
aver	689	2512	3175	2504	2510	2486	2522	391	9410
std	55	64	54	54	53	76	63	64	57

Table 5: Average execution time of the tasks with TOA (unit: 1 tick = 1/60 second).

The relative constraints are as follows:

$$\begin{pmatrix} s_3 \le f_1 \\ s_3 \le f_2 \\ s_8 \le f_1 \\ s_8 \le f_2 \\ s_4 \le f_3 \\ s_6 \le f_4 \\ s_5 \le f_8 \\ s_7 \le f_6 \end{pmatrix}$$

So we can write the matrix A as follows:

$$A^T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \end{bmatrix}$$

$$\chi^T = \begin{bmatrix} s_1 & f_1 & s_2 & f_2 & \cdots & s_8 & f_8 \end{bmatrix}$$

and

$$c^T = \begin{bmatrix} c_1 & c_2 & \cdots & c_8 \end{bmatrix}$$

Obviously, there exist four kinds of task dependencies: tasks dependencies between two tasks; sequential task dependencies among three tasks, cross dependencies among three tasks and extended task dependencies among three tasks. (No task dependencies among more than three tasks.) By calling lemma 4.1 and its extended approaches, we can prove that the deadlock situation in this example does not exist. Therefore, there exists at least one execution order of the eight tasks.

5.3.2 Step 2: Rules

- Before one task is loaded, all tasks on which it depends must be completed.
- Each task will be scheduled on the first available robot.
- The data dependency defines a partial ordering on process execution, e.g., T1 and T2 can execute in any order, but must be done before T3.
- All tasks must finish before the end of their periods.

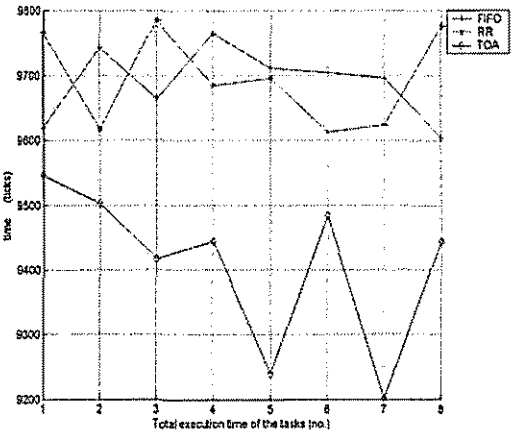

Figure 10: Experimental scheduling results with comparison of FIFO/RR/TOA

5.3.3 Step 3: Update execution orders

Once any task is done, an update process is triggered that updates the shared memory so that other processes such as the robots and the online scheduler can update their local variables. Therefore, even if the task has a different execution time because of time delay or any other reason, the scheduler can still guarantee the execution order (or task dependency) since it updates the global variable immediately after the task is finished.

5.3.4 Step 4: Stop

If all tasks are finished, stop; otherwise go back to Step 2 and continue. Table 5 gives the start time and average execution time of each task. From Table 5, we can see that all eight tasks are scheduled exactly as they are required by timing constraints and relative constraints. Due to the network time delays, the execution time of each task varies each time. But the execution order of the eight tasks is always the same, which means that with the proposed TOA, both the timing constraints and relative constraints can be satisfied despite different time delays. Furthermore, all the tasks are scheduled exactly after the father task/tasks are finished, which proves that the TOA is efficient even if all the execution orders are known without considering time delays. Fig. 10 compares the total execution time with regard to FIFO, RR and TOA. From Fig. 10, we can say that, FIFO and RR share the same features in scheduling a multi-robot system with both timing constraints and task dependencies, but TOA has a better performance in the total task execution time and is easy to schedule without considering the worst-case situation. The TOA can significantly improve the system performance with timing constraints and task dependencies.

6 CONCLUSIONS

A distributed multi-robot system may be used in various operations in robotics based industry tasks, for example, in tasks such as part assembly, auto painting, etc. This paper presents a task-oriented scheduling algorithm that considers both the timing constraints and relative dependencies among tasks. A matrix expression of the timing constraints and relative dependencies is presented that gives an

alternative approach for schedulability analysis. An offline scheduler is provided which can guarantee the schedulability and determine possible dead-lock situations.

Experimental results show that this algorithm can successfully solve task dependency problems while guaranteeing safety, task dependency and timing constraints. By comparing with the most commonly used scheduling algorithms in real-time operating system (FIFO and RR), TOA has a significant improvement in the total task execution time and is easy to implement.

7 ACKNOWLEDGMENTS

This research was supported in part by grants RGPIN227612 and RGPIN1345 from the Natural Sciences and Engineering Research Council (NSERC) of Canada and by the Canada Foundation for Innovation (CFI) under the New Opportunities Program.

References

[1] T. Arai, E. Pagello, L.E. Parker, "Guest editorial: advances in multirobot systems," *IEEE Trans. on Robotics and Automation*, Vol. 18, No. 5, pp. 655-661, Oct. 2002.

[2] J.C. Palencia and M.G. Harbour, "Schedulability analysis for tasks with static and dynamic offsets," *19th IEEE Systems Symposium (RTSS98)*, Madrid, Spain, pp. 26-37, 1998.

[3] A.K. Ramadorai, et.al, "Task definition, decoupling and redundancy resolution by nonlinear feedback in multi-robot object handling," *IEEE International Conference on Robotics and Automation*, Nice, France, pp. 467-474, 1992.

[4] X. Yun, "Coordination of two-arm pushing," *IEEE International Conference on Robotics and Automation*, Sacramento, CA, pp. 182-187, 1991.

[5] J.W.S. Liu, *Real-Time Systems*, Prentice Hall, New Jersey, 2000.

[6] M.C. Saksena, "Parametric scheduling for hard real-time systems," PhD thesis, 1993.

[7] A.J. Garvey and V.R. Lesser, "Design-to-time real-time scheduling," *IEEE Trans. on Systems. Man. and Cybernetics*, Vol. 33, No. 6, pp. 1491-1502, 1993.

[8] C.C. Han and K.J. Lin, "Scheduling distance constrained real-time tasks," *IEEE Real time Systems Symposium*, Phoenix, Arizona, pp. 300-308, 1992.

[9] J.F. Allen, "Maintaining knowledge about temporal intervals," *Comm. of the ACM*, Vol. 26, No. 11, pp. 832-843, 1983.

[10] S.M. Rinaldi, J.P. Peerenboom, T.K. Kelly, "Identifying, understanding, and analyzing critical infrastructure interdependencies," *IEEE Control Systems Magazine*, Vol. 21, No. 6, pp. 11-25, Dec. 2001.

[11] L. Abeni, G. Buttazzo, "Integrating Multimedia Applications in Hard Real-Time Systems," *The 19th IEEE Systems Symposium (RTSS98)*, Spain, pp. 4-13, Dec., 1998.

[12] B.P.-C. Yen and O.Q. Wu, "Internet scheduling environment with market-driven agents," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 34 , No. 2, pp. 281-289, 2004.

[13] M. Sato, "Gain-scheduled inverse system and filtering system without derivatives of scheduling parameters," *American Control Conference*, pp. 4173-4178, 2003.

[14] S.K. Baruah, J. Goossens, "Rate-monotonic scheduling on uniform multiprocessors," *IEEE Transactions on Computers*, Vol. 52 , No. 7, pp. 966-970, 2003.

[15] J.H. Kim, T.E. Lee, H.Y. Lee, D.B. Park, "Scheduling analysis of time-constrained dual-armed cluster tools," *IEEE Transactions on Semiconductor Manufacturing*, Vol. 16, No. 3, pp. 521-534, 2003.

[16] H. Darabi, M.A. Jafari, S.S. Manapure, "Finite automata decomposition for flexible manufacturing systems control and scheduling Systems," *IEEE Transactions on Man and Cybernetics*, Vol. 33, No. 2, pp. 168-175, 2003.

[17] H.M. Chaskar and U. Madhow, "Fair scheduling with tunable latency: a round-robin approach," *IEEE/ACM Transactions on Networking*, Vol. 11, No. 4, pp. 592-601, 2003.

[18] X. Zhu, S. Tu, "An improved dynamic scheduling algorithm for multiprocessor real-time systems , *Fourth International Conference on Parallel and Distributed Computing Applications and Technologies*, pp. 710-714, 2003.

[19] P. Yuan, M. Moallem, R.V. Patel, "A Real-Time Task-Oriented Scheduling Algorithm for Distributed Multi-Robot Systems," *IEEE International Conference on Robotics and Automation*, pp. 2562-2567, 2004.

[20] J. A. Stankovic, Chenyang Lu, S. H. Son, and G. Tao, "The case for feedback control real-time scheduling", In Proc. Euromicro Conference on Real-Time Systems, pp.11-20, 1999. 9.

[21] C. Lu, J. A. Stankovic, G. Tao, and S.H. Son, "Design and evaluation of feedback control EDF scheduling algorithm", In Proc. IEEE RTSS, pp.56-67, 1999.