Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2015, Article ID 273825, 14 pages
http://dx.doi.org/10.1155/2015/273825

Research Article

Multitasking Planning and Executing of Intelligent Vehicles for Restaurant Service by Networking

Jun Zhang,[1,2,3] Zhixian Chen,[1,2,3] Ying Hu,[1,2] Jianwei Zhang,[4] Zhenhua Luo,[1,2,5] and Xuehui Dong[1,2,5]

[1]*Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Xueyuan Avenue 1068, Shenzhen University Town, Shenzhen 518055, China*
[2]*The Chinese University of Hong Kong, Shatin, Hong Kong*
[3]*Shenzhen College of Advanced Technology, University of Chinese Academy of Sciences, Xueyuan Avenue 1068, Shenzhen University Town, Shenzhen 518055, China*
[4]*TAMS, University of Hamburg, Vogt-Kölln-Straße 30, 22527 Hamburg, Germany*
[5]*Shenzhen Graduate School, Harbin Institutes of Technology, HIT Campus, Shenzhen University Town, Shenzhen 518055, China*

Correspondence should be addressed to Ying Hu; ying.hu@siat.ac.cn and Jianwei Zhang; zhang@informatik.uni-hamburg.de

Received 19 December 2014; Accepted 2 March 2015

Academic Editor: Ching-Hsien Hsu

Indoor intelligent vehicles have been applied widely in restaurant service, where they are usually demanded to work for long period for multiple tasks and face the challenges of uncertainty, efficiency, and working online. In this paper, we propose an approach of multitasking planning and executing by networking for intelligent vehicles working for restaurant service. As to increase the efficiency of accomplishing multiple tasks, traditional HTN model is adapted to present the property of executing in parallel for the plan. As working online, the new inserted task is decomposed to get its individual plan, which is then merged into the global plan by optimization modelled as a maximal weight clique problem through evaluating the relations among actions and optimizing based on maximal cost saving first. Additionally, the protected states and goal states of an action are monitored in execution monitoring while action executing. Once exception occurs, the replanning based on partially backtracking would repair it. Moreover, with the mechanism of cooperation by networking, vehicles can allocate tasks, share information, and cooperate for execution monitoring. Finally, the test and demonstration of the experiment for drink service in restaurant environment show the feasibility and the improvement on the efficiency of multitasking.

1. Introduction

Today, indoor intelligent vehicles, sometimes also, namely, robots, have been applied more and more widely not only in logistics warehouses [1], but also in other public areas, that is, restaurant [2–4], official building [5], and hospitals [6], for delivery, surveillance, and information inquiry. Particularly in restaurant domains, shown in Figure 1, the vehicles are always demanded to deal with multitask in a long period, where multiple tasks are performed concurrently and efficiently.

Since the tasks may be too complex and the scale of the world is too large [7, 8], they may also work by distributed networking as multiagent system with heterogeneity, where each of them can work as an individual equipped with different capability for different job with others, and share information with each other via network communication. Additionally, the working environments are usually unconstructed and dynamic in these domains; in this condition, planning and executing for multitasking of the vehicles turns to be too complex under uncertainty. And there are two challenges to be taken up.

One is that the vehicles have to plan multiple tasks in the world with partially observable information. In an unconstructed and dynamic world, the vehicles may hardly acquire complete information required for tasks planning

Figure 1: Intelligent vehicle serving in restaurant.

with its own sensors; for example, to deliver a specified object, the location of the target object may be unknown since it is not inside the searching range or has been moved. Unfortunately, some unreachable information may be the key for the planning, and the planner cannot get a feasible and efficient plan for the task without it.

Another challenge is that the vehicles have to plan and execute online in dynamic world. In the real application, for example, restaurant, the tasks for the vehicles normally cannot be preset in advance, and they may come stochastically at any time. When a new task comes, the vehicle may be working for other tasks; at this time, the new task has to be inserted into the current task list and treated properly to get a more efficient plan rather than waits until the current tasks are finished. Additionally, since the states of world may change dynamically by external events, exception may rise and lead to failure of the task when executing the plan. To recover the failure, vehicles usually have to reconfigure its status, make a new plan, and maintain it online.

In the literatures, many studies have put their effect to rise to the challenges. On the one side, to figure out the uncertainty, previous works have shown their methods in information gathering, planning with incomplete information [9, 10]. Moreover, Markov decision processes (MDP) [11] and its expansion partially observable MDP (POMDP) [12, 13] are applied to predict the states of the world and make decision during planning. In [14], Kaelbling and Lozano-Pérez described an integrated strategy for task and motion planning based on planning in belief space and showed its robustness and flexibility in the mobile manipulation problems with substantial uncertainty. On the other side, for planning and executing online, the methods of execution monitoring and replanning have been developed in [15, 16]. As an alternative view, continuous planning [17] provides a practical approach to planning online in dynamic world [18]. In [19, 20], an approach based on continual planning was proposed to enable the agents planning and acting in dynamic environment, where the concept of assertions is introduced to describe why and when an agent should switch between planning and acting.

Indeed, probability theory minimizes the uncertainty of the world, and the mechanism of both execution monitoring and continual planning guarantees the achievement of the task. However, the probability approaches are with high computational cost. In an alternative opinion, if the future always cannot be predicted precisely by applying probability inference, then the simplified way is to make the uncertainty under the monitoring of sensors.

Thanks to the development of the sensor networking, sensor communication technology has been employed for intelligent vehicles. In [21–23], the authors provided the low-cost solutions for indoor localization, especially for colocalization in a networking environment. In these studies, sensor networking plays an efficient role in extending the reachability and enhancing the robustness of the vehicles. However, in these methods, the sensor network has to be built by deploying the sensors distributed in the work environment which would be a tough work in an unconstructed world.

Instead of building a distributed sensor network in the environment, cooperating in multiagent system provides an alternative view to plan under uncertainty. Micalizio and Torasso presented a distributed approach for monitoring and diagnosing the execution of a multiagent plan in partially observable environment [24]. However, the approach mostly aimed at recovering the exception of plan execution rather than preventing the failure of planning.

In this paper, aiming for multitasking problem of intelligent vehicles in restaurant environment, it is assumed that the vehicles can share information in a multiple vehicles network with heterogeneity, and the main idea is to handle the uncertainty and planning online through the cooperation of networking. Based on this idea, we propose a novel approach to enhance the robustness and improve the efficiency of multitasking by integrating multitasking planning, executing, and networking. The features of the approach and our contributions are summarized as the following:

(i) extending the model of HTN planning for multitasking planning under uncertainty and adapting the decomposing algorithm for the extended model,

(ii) developing the optimizer for parallelizing plans for multiple tasks, in which the actions optimization is modelled as a maximum weighted clique problem by evaluating the relations among actions and searching the action clique with maximal cost saving,

(iii) Presenting the method for integrating executing, monitoring, and replanning in dynamic environment, with which the execution can be monitored with the protected states, which must be kept from changing during the action execution, and goal states. In addition, the algorithm of replanning based on partially backtracking is developed.

The paper is structured as follows: Section 2 introduces the overall approach. In Section 3, task decomposing is introduced. Followed with Section 4, the optimization for plan parallelizing is presented. After that, in Section 5, the method for plan executing and monitoring is introduced. At last but not least, the experiments for restaurant service are demonstrated and the result is discussed in Section 6 and the conclusion and future work are summarized in Section 7.

2. Overall Approach

2.1. Control Architecture. The design of the control architecture for intelligent vehicles networking has a very important

impact on the robustness of the system, and several different architectures have been described in [25]. However, different from most traditional multiagent systems, in this study, each vehicle in the network is supposed to be responsible for particular tasks and equipped with specified capabilities different from others. At the meantime, each vehicle is expected to have more authorization of local control so that new team members are facilitated to join in network. For example, in an automated restaurant, there are two vehicles or more; particularly, one is a waiter working for menu service, another one is for delivering food and clearing tables, and if there are others, they may provide reception service, or even others. Each vehicle may take on responsibility for different jobs in serving for guests and work as individual with high autonomy, as well as communicating with others when necessary.

To realize both robustness and flexibility of the system, as a hybrid architecture, the distributed robot architecture (DIRA) [26] is adapted to our system. Besides allowing autonomy in individual, the DIRA also facilitates explicit coordination among robots and employs the layered architectures which are popular with single vehicle systems. Based on DIRA, the control architecture is shown in Figure 2.

In this architecture, intelligent vehicles connected in network can interact with each other via their own networking interface, with which cooperation requests are published and subscribed, and each vehicle has a complete layered architecture with a loop of planning, executing, and monitoring on task level (see in the blue dashed line). Taking the vehicle for delivery service as an example, the vehicle works online and waits for recalling. Once a new task is generated by task service and inserted to task list, five processing phases are involved to accomplish it.

In task decomposing (TD) phase, the initial states will be checked whether there is complete information for planning. Due to the partially observability of the world, information may be incomplete and undetermined. In this case, the vehicle hands out a request of state acquisition to the interface, where the request is published to the network. Then, the undetermined states would be acquired by the cooperation from other vehicles. If the known states are sufficient for planning, TD is recalled to decompose the inserted task by using hierarchical task net (HTN) and its individual plan is obtained.

In plan parallelizing (PP) phase, the individual plan of the inserted task is merged into the global plan for all processing tasks, and the actions from each individual plans will be paralleled for multitasking. However, plan parallelizing will process only one layer each time and generate a combined action. Only if the combined action is successfully executed or new task is inserted, the parallelization will go ahead to next layer. This will save the computation consumption and increase the real-time performance.

In plan executing (PE) phase, the combined action in the global plan will be performed in parallel, where the actions are transformed into executing commands and sent to the controllers; at the same time, the protected states and the goal states are delivered to the execution monitoring.

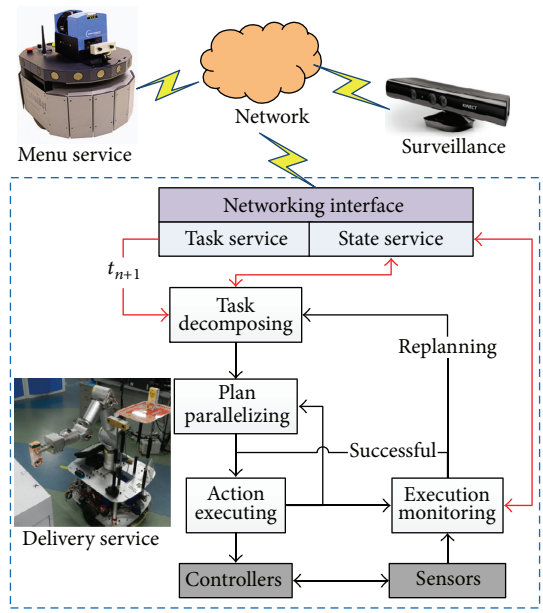


Figure 2: The control architecture.

In execution monitoring (EM) phase, the protected states and goal states will be monitored on semantic level with the sensors. Likewise, those states are also published to the state service, so that other vehicles can cooperate for monitoring them. If there is no destroyed protected state and all goal states of an action are achieved, the action execution is successful and PE is recalled to execute next actions. Otherwise, if either any protected state is destroyed, or the goal states cannot be achieved in regular time, the execution fails and the replanning tries to repair it. As to increase the efficiency for replanning, a partially backtracking algorithm is applied to minimize the cost of replanning.

Furthermore, the controllers and the sensors together constitute a route of reactive control for action execution in behavior layer. The controllers include the mobile control, robot arm control, and pan-unit control, while the sensors include the laser, binocular vision, camera, and gyroscope. And the geometric planners, including motion planning and path planning, are supposed to be implemented in behavior layer.

2.2. Cooperating by Networking. The cooperation among intelligent vehicles by networking includes two capacities: task service and state service, as shown in Figure 2. There, task service allocates the tasks for the vehicles in network, while the state service provides the sharing states and publishes the request of state acquiring and monitoring.

For long term service, the vehicles in a restaurant have to work online. Through task service, a vehicle can publish new tasks to other vehicles as well as receiving new tasks published from others. For example, when the vehicle for menu service gets an order from guest, then it will publish a new task to task service, and others will receive and check whether the new task is suited for performing. If the new task is suited for

a vehicle, it will be inserted to the task list of this vehicle, and others will ignore it.

On the other hand, as discussed previously, the world the vehicles are working in is usually under uncertainty and dynamic, so that the information for TD and EM is often incomplete for individual vehicle. To handle this problem, the state service enables the cooperation among the vehicles in network to share new states and publish request of state acquiring. On one side, in TD, before the task is decomposed, the current states have to be checked whether they are sufficient to decompose the task; if there is lack of information, the request of acquiring this information is published to state service and others can help to complete this information; once the information is obtained, it will be shared to state service, and the vehicle that needs it will be able to continue working. On the other side, in EM, during execution monitoring, the protected states and goal states have to be monitored online. For an individual vehicle, to monitor the change of the environment and execute an action at the same time, sometimes, it is tough and inefficient. In this situation, with the state service, other vehicles can be involved to monitor these states.

2.3. Action Knowledge of Restaurant Domain. To model the domain of restaurant service, the HTN model [11] is expanded and the SHOP2 syntax [27] is employed to described the knowledge of the restaurant domain. As one of the most famous planning method, HTN model provides a syntax of defining the methods by controlling the decomposing of tasks and avoiding the irrelevant searching branches, and this speeds up significantly the task decomposition [28]. However, for cooperating and monitoring of the execution, the action knowledge should be able to present how to cooperate and what to monitor, and it can be formalized as planning domain as follows.

Definition 1 (planning domain). A planning domain D is a tuple $D = (S, O, M)$, where $S = (s_0, s_1, s_2, \ldots)$, $O = (o_0, o_1, o_2, \ldots)$, and $M = (m_0, m_1, m_2, \ldots)$ are the limited sets, respectively, of states, operators, and methods.

Similar with the traditional definition, planning domain includes the states, operators, and methods. And the three elements are defined, respectively, as Definitions 2–4.

Definition 2 (state). A state s is an atomic sentence as the form of (*predicate*, *term*, etc.) which describes a fact of the world.

Even though the traditional HTN model is weak for handling uncertainty by assuming the knowledge about the working world is complete, we keep this property to guarantee the high performance on planning efficiency and figure out the uncertainty through an exertional modular of state acquisition, detailed in the following section. Different from the belief state, the determined state indicates a fact. For instance, (*at*, *vehicle1*, *waypoint1*) indicates that a vehicle, namely, *vehicle1* is at a way point namely *waypoint1*, and (*vehicle*, *vehicle1*) indicates that *vehicle1* is a vehicle.

Definition 3 (operator). An operator o is a 5-tuple

$$o = \big(\text{name}(o), \text{precond}(o), \text{add}(o), \text{delete}(o), \text{protected}(o)\big), \quad (1)$$

in which name(o), precond(o), add(o), delete(o), and protected(o) are, respectively, the set of the name, preconditions, positive effects, negative effects, and protected states.

As the introduction of what should be monitored during action execution, the protected(o) describes the states that have to be protected and cannot be changed while executing the action. In Box 1, an example of operator (*!move_base*, *?from*, *?to*) is given. There, the head contains the name *!move_base* and parameters "*?from*" and "*?to*" which is with a question symbol to distinguish from instance. The other elements of operator are listed follow the head, particularly, the protected states (*available*, *?to*) and (*passable*, *?from*, *?to*) indicate that to guarantee the execution of (*!move_base*, *?from*, *?to*), the target location *?to* must be available and the way between *?from* and *?to* must be passable during the execution.

Definition 4 (method). A method m is a 4-tuple

$$m = \big(\text{name}(m), \text{task}(m), \text{precond}(m), \text{subtask}(m)\big), \quad (2)$$

in which name(m) is the name of the method, task(m) is the generalization of nonprimitive task, precond(m) is a set of literals for the preconditions of the method, and subtasks(m) are a set of ordered subtasks of m.

A method describes the prescription of decomposing a nonprimitive task into subtasks. And an example method, namely (*serve_drink*, *?drink ?guest*), is defined in Box 2.

In Figure 3, the plan hierarchy for the task of *serve_drink*(*drink1*, *guest1*) is shown, in which 5 methods and 2 operators are involved. There, the red arrows indicate the executing order for the subtasks in the same layer, and the grown boxes present the protected states of respective primitive action.

3. Task Decomposing

3.1. Problem Modelling. Similar to traditional HTN, a planning problem for multitasking is defined as Definition 5.

Definition 5 (planning problem). A planning problem P is modelled as a 3-tuple

$$P = (D, S_0, T), \quad (3)$$

in which D is the planning domain, $S_0 = (s_1, s_2, \ldots, s_n)$ is a set of initial states, and $T = (t_1, t_2, \ldots, t_n)$ is a list of target tasks.

A planning problem provides the action knowledge, initial states, and the target tasks for the planner, and with them, the solution will be found, if there is one. In traditional HTN planning, the solution is defined as plan $\pi = \langle a_0, a_1, a_2, \ldots \rangle$, where a_i is a primitive action and π is a sequence of total ordered actions. In the definition, the actions in π can only be

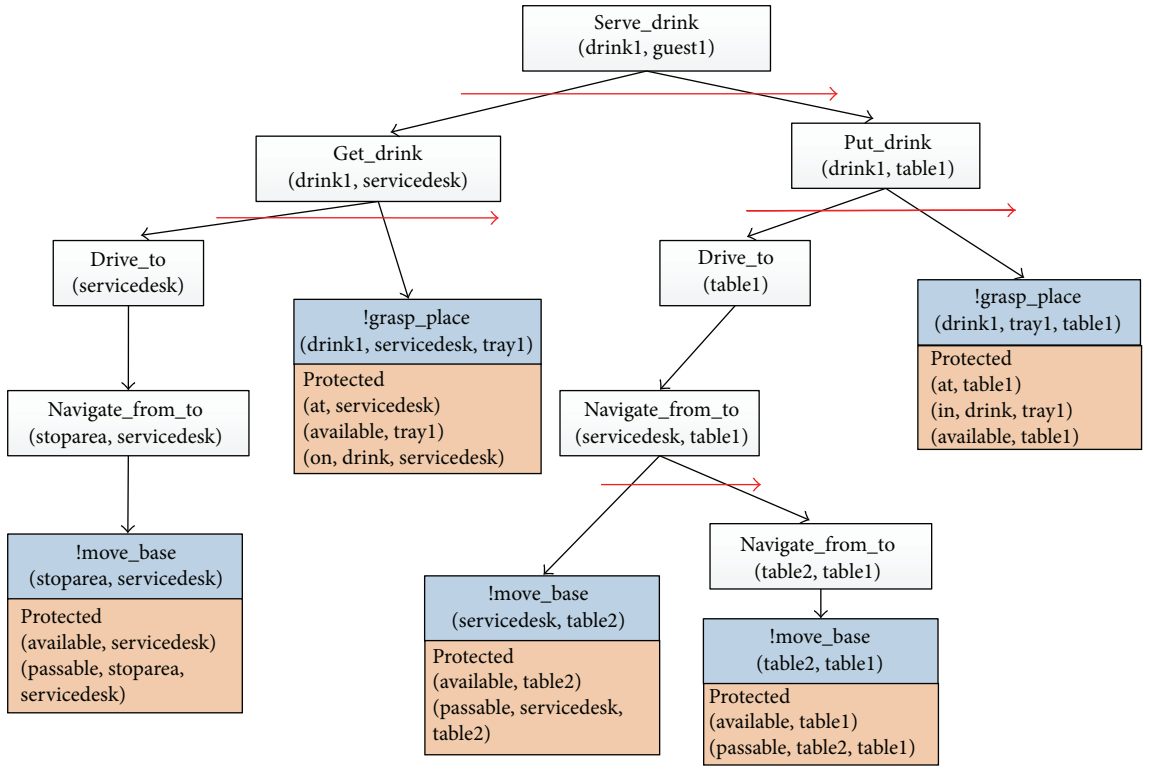


Figure 3: Plan hierarchy for serving a drink.

```
(:operator (!move_base, ?from, ?to)
(   ; precondition
    ((location, ?from), (location, ?to), (passable, ?from, ?to),
    (at_vehicle, ?from), (available, ?to))
    ;delete
    ((at_vehicle, ?from), (available, ?to))
    ;add
    ((at_vehicle, ?to), (available, ?from))
    ;protected
    ((available, ?to), (passable, ?from, ?to))
)
```

Box 1: An example of operator.

```
(:method (serve_drink, ?drink ?guest)
(   ; preconditions
    (drink, ?drink), (guest, ?guest),
    (at, ?guest, ?table1), (on, ?drink, ?table2)
    ; subtasks
    (get_drink, ?drink, ?table2), (put_drink, ?drink, ?table1)]
)
```

Box 2: An example of method.

executed one by one corresponding with the orders. However, for multitasking, it cannot represent the situation of executing actions in parallel. To provide a plan feasible for multitasking, a concept of action clique and the new definition of plan are introduced.

Definition 6 (action clique). An action clique is a pair

$$\mathrm{AC} = (A, C), \tag{4}$$

in which $A = \{a_1, a_2, \ldots, a_i\}$ is a set of actions a_i partially ordered and C is a set of constraints among actions.

Definition 7 (plan). A plan is

$$\pi = \langle \mathrm{AC}_1, \mathrm{AC}_2, \ldots, \mathrm{AC}_n \rangle, \tag{5}$$

which is a total ordering sequence of action clique AC_i.

A plan is a hybrid construct, in which actions are ordered in two layers. In the plan layer, the action cliques are total ordered; there, an action clique AC_i is available only if the preconditions of all actions inside are satisfied after the performance of AC_{i-1}. However, in clique layer, actions are partial ordered and can be executed in parallel.

In the whole planning process, the plan will appear in two different forms; to store and reuse them, the task tree and plan tree are defined.

Definition 8 (task tree). A task tree, TaskTree(T_i), is a hierarchical tree presenting the decomposing architecture of task T_i.

A task tree shows how a task is decomposed into subtasks recursively until the primitive tasks are achieved. There, each node stands for task which may be a primitive task or a nonprimitive task, and the children nodes are its subtasks.

Definition 9 (plan tree). A plan tree, PlanTree(P), is a reachability tree representing the action paths to achieve the goals of the tasks.

A plan tree shows how to accomplish the tasks by executing the plan. In a plan tree, each node is an action or action clique. Normally, the initial plan tree contains multiple branches, and each branch, from the root to the leaf, is an action path to achieve the goals of respective tasks. Since the plans will be paralleled layer by layer in plans parallelizing, the actions distributed in each branch will be insisted into action cliques by optimizing. Finally, if the tree shrinks into a single branch, it is the final plan tree for all tasks.

3.2. Algorithm for Task Decomposing. The goal of TD is to generate the individual task tree and the individual plan tree for the tasks. As initialization, a global task tree rooted as "*TaskTree*" is created to store the task tree for each independent task, which is attached as the subtree of "*TaskTree*." Similarly, a global plan tree rooted as "*PlanTree*" is created, and every plan tree of each independent task will be attached to the current action clique as a new child tree.

To plan online, the planner listens to task service and waits for new task insertion. Once a new task t_n is inserted, it will be the initial member of *tasklist* and decomposed independently to get the task tree and plan tree by the function of *decompose_task*(), shown in Algorithm 1.

The *decompose_task*() performs a recursive process to decompose the inserted task. At the beginning, the ending condition is checked, and once the *tasklist* is empty, it returns the *tasktree* and the *plantree*. Otherwise, the *activetask*, which is always the first element of *tasklist*, is treated in two ways. (1) If *activetask* is an instance of operators, a circle block repeats the acquiring of sharing state and check precondition. There, *check_precond*() checks whether all preconditions of *activetask* are ready in renewed S_0. If not, it publishes the preconditions which are lacking to state service; or else, the preconditions are sufficient; it breaks the circle. And then S_0 is renewed as S' by applying *activetask*. And if S' is not *false*, *activetask* is added to *plantree*, and *decompose_task*() is recalled to decompose next task in *tasklist* under S'. (2) If *activetask* is an instance of methods, the function *check_precond*() is implemented as it does in the first condition. While the preconditions of *activetask* are met, the *subtasks of activetask* will be added to the *node of activetask* as its children and inserted to the front of *tasklist* after deleting *activetask* from *tasklist*; after that, the *decompose_task*() is recalled with the new *tasklist*. And this process will be implemented until the ending condition is satisfied.

4. Plan Parallelizing

4.1. Process of Plan Parallelizing. The objective of PP is to parallelize all the individual plans for target tasks, so that the tasks can be executed more efficiently in parallel, and the process is shown in Figure 4, and the subprocesses are listed as follows.

Description of the Subprocesses

Tr_1: new task is inserted and decomposed.

Tr_2: the active AC is executed successfully.

Tr_3: the execution for the active AC fails.

To_1: append the new plan tree to global plan tree.

To_2: activate the next layer of the global plan tree.

To_3: replace the failing branches with repaired plan tree.

To_4: insert the new AC and delete the actions of AC from respective branches.

There, PP will be activated by three triggers and its respective operations of plan tree; after activating the new layer, the relations among the actions in active layer, as well as the relations between the actions and the executed AC, are analyzed in relation analyzing. With the relations, the optimization is performed to select actions into new AC, and then it will be inserted into the global plan tree by To_4.

In the following, we introduce the details for relation evaluating and actions optimizing.

4.2. Action Relations. As the definition of action clique, the actions in an action clique must be compatible for execution in parallel; hence, the relations among actions must be evaluated. In literatures, the analyzing technique of action relations has been introduced in Graphplan [29] and LCGP [30]; inspired by these works, three relations are defined as follows.

Action Relations

Relation 1 (independence). Consider

$$\begin{aligned}&\left\{\mathrm{del}\left(a_i\right) \cap \left[\mathrm{pre}\left(a_j\right) \cup \mathrm{add}\left(a_j\right)\right]\right\}\\ &\quad \cup \left\{\mathrm{del}\left(a_j\right) \cap \left[\mathrm{pre}\left(a_i\right) \cup \mathrm{add}\left(a_i\right)\right]\right\} = \emptyset.\end{aligned} \tag{6}$$

Relation 2 (allowance). Consider

$$\begin{aligned}&\left[\mathrm{del}\left(a_i\right) \cap \mathrm{pre}\left(a_j\right)\right] \cup \left[\mathrm{add}\left(a_i\right) \cap \mathrm{del}\left(a_j\right)\right] = \emptyset,\\ &\left[\mathrm{pre}\left(a_i\right) \cap \mathrm{del}\left(a_j\right)\right] \cup \left[\mathrm{del}\left(a_i\right) \cap \mathrm{add}\left(a_j\right)\right] \neq \emptyset.\end{aligned} \tag{7}$$

Relation 3 (retraction). Consider

$$\mathrm{del}\left(a_i\right) = \mathrm{add}\left(a_j\right) \quad \text{or} \quad \mathrm{add}\left(a_i\right) = \mathrm{del}\left(a_j\right). \tag{8}$$

The relations are defined by the interplay between actions as the equations in Action Relations. As the definition

```
decompose_task(S0, tasklist, tasktree, plantree)
    if tasklist is empty:
        return(tasktree, plantree)
    activetask ← tasklist(0)
    if activetask is an instance of operator:
        while 1:
            get sharing states and renew S0
            check_precond(S0, activetask.precond)
            if there are preconditions not ready:
                publish the preconditions which are not ready
            else: break
        S' ← γ(S0, activetask)
        if S' is not False:
            plantree.add_child(activetask)
            tasklist.remove(activetask)
            tasktree, plantree ← decompose_task(S', tasklist, tasktree, plantree)
    if activetask is an instance of method:
        while 1:
            get sharing states and renew S0
            check_precond(S0, activetask.precond)
            if there are preconditions not ready:
                publish the preconditions which are not ready
            else: break
        subtasks ← method(S0, activetask).subtask
        tasklist.remove(activetask)
        tasklist ← insert subtasks in the front of tasklist
        tasktree.add_children(subtasks)
        tasktree, plantree ← decompose_task(S0, tasklist, tasktree, plantree)
```

Algorithm 1: Decomposing task based on hierarchical decomposing.

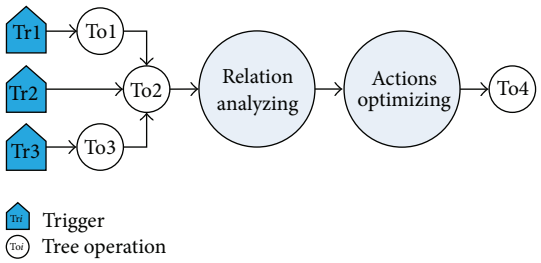


Figure 4: Process of plan parallelizing.

shown, the relation between actions is dependent on their precondition resource and output effect. There, a_i and a_j are independent with each other while the performance of a_i does not delete the preconditions and positive effects of a_j, and at the meantime, the performance of a_j does not delete those of a_i either. And if the performance of a_i will neither delete the preconditions of a_j nor produce the negative effects of a_j and, at the meantime, the performance of a_j will either delete the preconditions of a_i or produce the negative effects of a_i, we say a_i allows a_j. Furthermore, a retraction relation between a_i and a_j is met while the positive effects of a_i is equal to the negative effects of a_j, or vice versa.

The criteria to justify whether certain actions or tasks are parallelizable can be summarized as two relations: independence and allowance. However, the independence is more restricted than the allowance. Different from the definition in LCGP, our definition of allowance is more strongly constrained and divided into two cases: a_i allows a_j and a_j allows a_i. But in LCGP, the allowance means only a_i and a_j are with allowance relation, and this definition includes the independence. The advance of our criteria is that the actions with allowance relation are ordered and are more appropriate for actions execution.

There, the independence and allowance relations are checked among the actions in the same layer of the plan tree, while the retraction relation is checked between an action in current layer and the actions in the latest action clique. And the respective rules for assembling them in an action clique are listed as follows.

(i) If actions are independent of each other, they can be put together in an action clique and executed simultaneously.

(ii) If two actions a_i and a_j are with allowance relation, they can be put in the same action clique and in case of independence relation, but with an allowance order, which indicates that a_i can be executed before a_j if a_i allows a_j.

(iii) If there is any action in active layer with a retroaction relation, it should be excluded from the candidates for the optimization.

Besides, actions are not with any one of above relations; they are mutually exclusive and cannot appear in the same action clique.

4.3. Optimizing as Maximum Weight Clique Problem. After the preselection by analysing the relations among actions, the actions with independence and allowance relations will be optimized. And we model the optimization as a MWCP (maximum weight clique problem) [31, 32], as

$$G = (A, W, E), \tag{9}$$

where

(i) $A = \{a_1, a_2, \ldots, a_n\}$ is a set of vertices representing action candidates for action clique,

(ii) $W = \{w_{a_i} \mid a_i \in A\}$ is a set of combined weights w_{a_i} valued with the time cost for the execution of the according action and the number of times that this action has shown in current layer,

(iii) $E = \{(a_i, a_j) \mid a_i, a_j \in A, i \neq j\}$ is a set of edges connecting the relative actions a_i and a_j.

In a MWCP, the vertices are the actions with independence relation or allowance relation in the active layer of plan tree. An edge indicates the relation between two connected actions, where an undirected edge presents an independence relation and a directed edge presents an allowance relation. An example graph of MWCP is shown in Figure 5, where the vertices are named with actions and their weights are noted on the nodes. In the example, one independence and six allowances are included.

Since the goal of optimization is to find an action clique to reduce the time consumption for multitasking and the global optimization is always complicated and needs a huge consumption of computation, the second best thing is to find the local optimum. As this, the rule of the optimization is set as maximal-cost-saving-first compared with the total time cost to execute all actions involved in the action clique one by one, and the formula is as (10), where n_{a_i} is the amount of that a_i appearing in the active layer of plan tree, and $W(c_x)$ is the cost of action clique c_x:

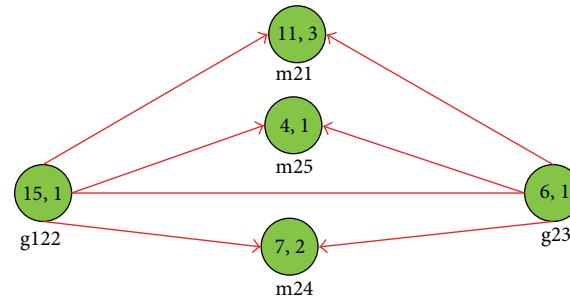
$$f_{\max}\left(w_{a_i}, c_x\right) = \max\left(\sum_{c_x}\left(w_{a_i} * n_{a_i} \mid a_i \in c_x\right) - W\left(c_x\right)\right). \tag{10}$$

However, since an action clique may contain allowance relations, the actions in clique have to be sorted by allowance order. And $W(c_x)$ is equal to the maximum weight for all action paths in c_x, shown in

$$W\left(c_x\right) = \max\left(\sum_{ap_k}\left(w_{a_i} \mid a_i \in ap_k\right) \mid ap_k \in c_x\right). \tag{11}$$

In particular, if there is neither independence nor allowance relation among actions in the active layer, the optimization is simplified as

$$f_{\max}\left(w_{a_i}, a_i\right) = \max\left(w_{a_i} * \left(n_{a_i} - 1\right) \mid a_i \in A\right). \tag{12}$$



m21: (move_base, table2, table1)
m25: (move_base, table2, table5)
m24: (move_base, table2, table4)
g122: (grasp_place, drink1, tray2, table2)
g232: (grasp_place, drink2, tray3, table2)

Figure 5: An example graph of MWCP.

As the rule of optimization, the selected actions clique of the example in Figure 5 will be AC = {($g122$, $g232$, $m21$), independence ($g122$, $g232$), allowance ($g122$, $m21$), allowance ($g232$, $m21$)}.

5. Plan Executing and Execution Monitoring

5.1. Plan Executing. After a planned action clique is generated by plan parallelizing, it is executed in PE. As an action clique consisted of the actions executable in parallel, PE should play two roles during the execution: (1) to manage and allocate the actions in action clique and (2) to transform the actions into executing command.

Algorithm 2 shows the main algorithm for executing plan. There, PE works online and waits for the trigger until a new AC is generated in PP through *wait_for_ac*(). Once an AC is got, the actions in AC will be executed iteratively until it is empty. Since the actions in an AC is executable in parallel but with allowance constrain, in which an action can only be executed immediately when there is no prefix action of it, all the free actions, without prefix action for them, are transformed to controllers, respectively, by *tramsform_to_control* ($action_i$) at once. At the meantime, their protected states and effected states, including the positive effect and negative effect, are published to EM by *publish_to_monitor* ($action_i$) and then the free actions will be removed from current AC as well.

In *tramsform_to_control* ($action_i$), $action_i$ is transformed to control function by recalling the respective controller. For example, an action *!move_base* (*Table1, Table2*) is to move the vehicle from *Table1* to *Table2*, where *Table1* and *Table2* will be mapping to the corresponding positions in the map and assigned to the navigation function as its parameters, in which path planning is called to find a path way and drive the vehicle from the position of *Table1* to the position of *Table2*. Likewise, the protected states of $action_i$ are published to EM, where they will be monitored.

After transforming the free actions to control, PE will wait for the response of EM. Once an action is successfully executed, it is returned as *consummator* and removed from the prefix list of other actions to free them for executing

```
execute_plan()
    AC ← wait_for_ac()
    while AC is not empty:
        for each action_i in AC:
          if prefix_of(action_i) is empty:
              tramsform_to_control(action_i)
              publish_to_monitor(action_i)
              AC.remove(action_i)
          consummator ← wait_for_monitor()
        for each action_i in AC:
          if consummator in prefix_of(action_i):
              remove consummator from prefix_of(action_i)
    activate PP to parallel next layer
```

Algorithm 2: Plan executing.

immediately. When all actions in AC are executed, PE will send a trigger to activate PP to parallel next layer of the global plan tree.

5.2. Execution Monitoring. In EM, the plan execution is monitored by two ways: (1) the protected states have to be kept and never be tampered until the execution is finished; (2) the goal states are expected to be realized, where the positive effect states are shown in current states and the negative effected states are deleted from current states.

Each state of protected states and goal states is monitored by a process of the involved sensor, in which the sensor acquires the information and confirms the states. EM manages the processes by calling or killing them and compares the confirmed states with the monitoring states of each action. There are three cases.

(i) If any protected state of an action has been confirmed to be tampered during executing the action, the execution fails, and the relative plan must be replanned.

(ii) If the goal states are all confirmed to be achieved and the protected states are all kept well during the execution of an action, then the action is executed successfully. At this time, the action is sent to PE as a consummator, and this would activate the process of PE to go ahead to execute the next batch of free actions.

(iii) If the goal states cannot be confirmed in expected time, then the execution of this action would be considered as failure, and it will be replanned also.

While the execution fails, the replanning is activated to recover it. As to improve the performance, an algorithm based on partially backtracking is designed to restrict the replanning depth. As the basic principle, the replanning always and iteratively tries to replan the nearest upper task of the failure action or task until the new solution is obtained.

As shown in Algorithm 3, the corresponding action in task tree can be achieved by *Action2Task*(), in which a dictionary maps the actions between plan tree and task tree. The parent task is then replanned. If the replanning is successful, the *repairedplan* is got and replaces the failure one waiting for next process of plan parallelizing by *renew_plantree*(), and the *repairedtask* repairs the failure task tree by *renew_tasktree*(). If not, the upper parent task is got and replanned recursively until solution is found or the root is reached with returning false.

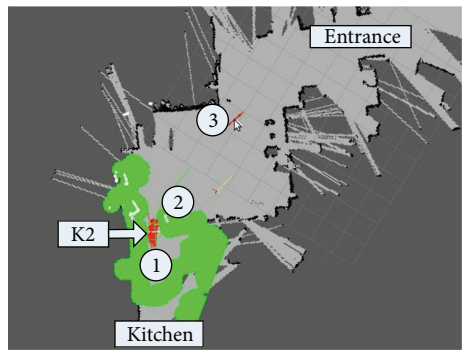


Figure 6: Map for the experiment.

6. Experiment and Result

6.1. Experiment Environment and Setting. The experiment is tested in a dynamic environment setting with one entrance and three tables distributed in a laboratory, shown in Figure 6, which simulates the scenario of drink service in automated restaurant. Three facilities, two intelligent vehicles, and settled Kinect are involved in the networking of the experiment; each of them has the individual responsibility, but cooperating with each other for acquiring information and monitoring execution.

For details, each facility has the following capability: (1) K1, Kinect for surveillance settled at the door way of the entrance, can detect and recognize the guest who enters the service area or goes out from the entrance; (2) K2, a pioneer intelligent vehicle equipped with Kinect for menu service, can detect and recognize which table the identified guest is sitting by and get his or her order for drink; (3) K3, another intelligent vehicle equipped a 7 DOF arm and a gripper for deliver service, can serve the ordered drink to the corresponding table and recycle the empty drink boxes after guests leave.

To cooperating in network, due to the convenience for communicating among distributed machines in local area network [33], ROS (robot operating system) is employed for the development of control systems. In addition, to enhance the robustness of the networking, we construct a hybrid network environment by using wireless data transmission modules as the supplement. In the experiment, communication among all facilities is through one shared "*roscore*" in ROS, shown as Figure 7, while K2 and K3 communicate through wireless data transmission.

In Figure 8, we can see the flowchart of multiple sensors. On the left top, K1's and K2's computers are connected in the same local area network; among them, K2's laptop is running as a ROS master, and other computers are running connect

```
repaire_plan(S, fAction, preAction)
    fTask ← Action2Task(fAction)
    solution ← False
    while solution = False:
        fParent ← fTask.up
        if fParent is the root of TaskTree:
            return False
        repairedtask, repairedplan ← decompose_task(S, fParent, repairedtask, repairedplan)
        if repairedtask, repairedplan is not False:
            solution ← True
    renew_tasktree(repairedtask)
    renew_plantree(repairedplan)
```

Algorithm 3: Repair the failure plan by replanning based on partial backtracking.

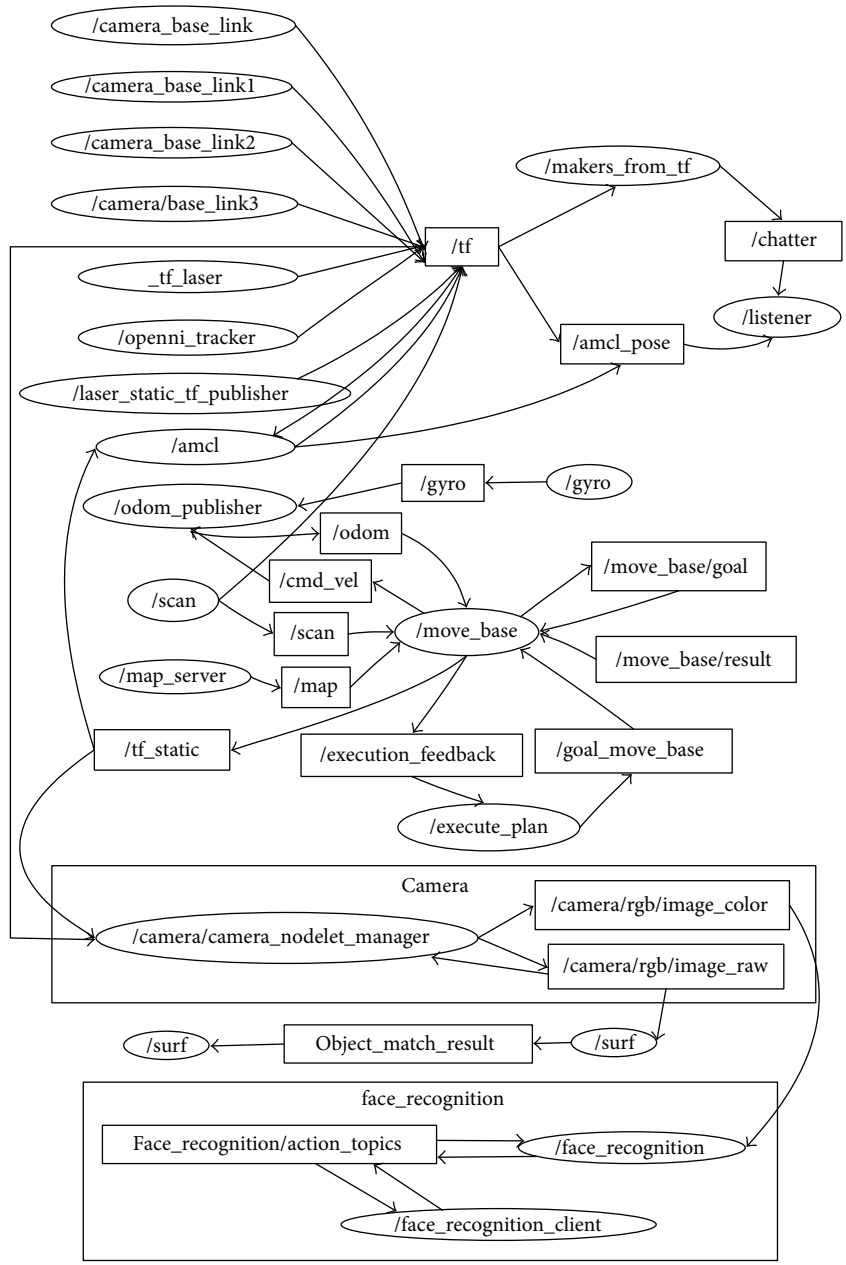


Figure 7: ROS computation graph for K2.

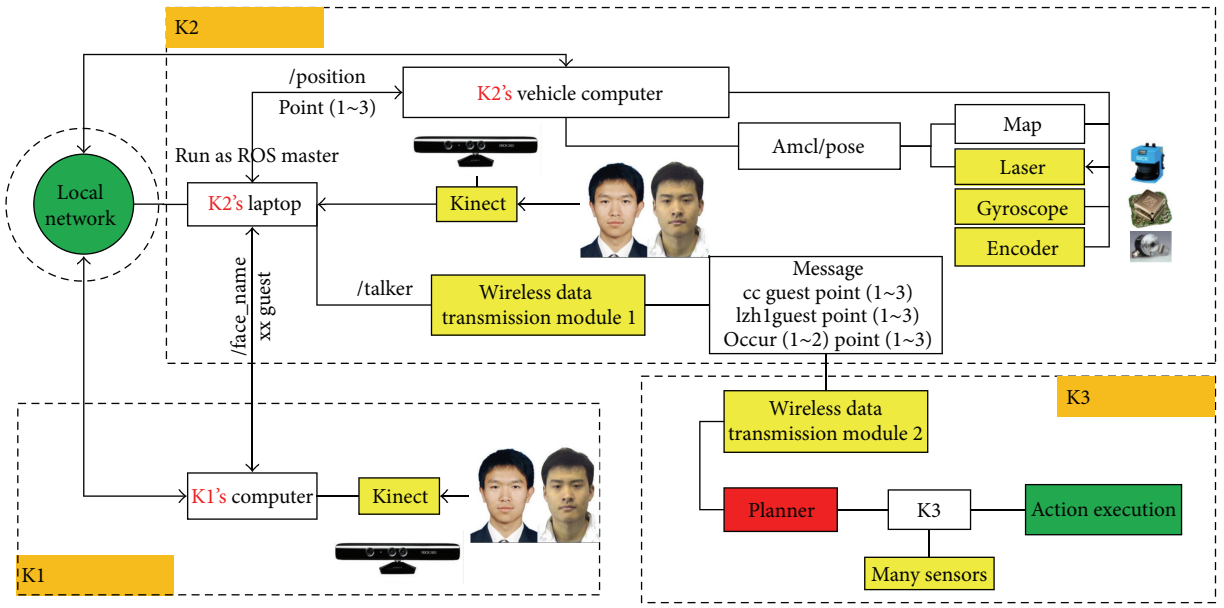


Figure 8: The flowchart of multiple sensors.

with it, so that the K1's and K2's computers can share ROS topic with each other. They are assembled as a group. K1 sends topic of "*/face_name*" to the group by using its Kinect sensor to detect guests and record their ID. And then K2's Kinect sensor can detect guests and recognizes their names, at the same time; K2's vehicle sensors send back a topic of "*/position*" to the group. Using all these topics, K2's laptop can know the guests and their sitting tables; then it sends a topic message using K2's wireless data transmission module to K3. K3's planner receives the messages and works with them.

In experiment, K1 monitors the entrance to detect and acquire training images for every new guest. Human detection is realized by the skeleton tracking, which is the best known application of Kinect, and the joint positions of a person standing in front of the camera can be detected and tracked through using the ROS package of *openni_tracker* [34]. If a new guest stares at the camera for couple of seconds, the desired number of training images is obtained from the video stream, and the ID for the new guest is then assigned [35]. Finally, the acquired images are stored and added to the list of training imaged and shared with others. The process is shown in the images of Figure 9.

As a pioneer vehicle, K2 is capable of SLAM (simultaneous localization and mapping), communicating with K1 by "*roscore*" and exchanging information with K3 by wireless data transmissions. Equipped with the navigation stack of ROS, it is able to plan path, create a map from laser scan data, locate itself in a map, and so forth. In Figure 9, K2 starts to routine patrol to detect and recognize every guest (as shown in Figure 10), confirm their table number through its current location and pose, acquire the orders of drink, and publish them to K3. If the K2 detects the drink only but no person is near the table, it will infer whether the guest has gone by the K1's information.

Figure 9: *Guest lzh1* is detected and recognized by K1.

Moreover, as to make the experiment easy going, but not the key point of our experiment, we design SURF detector to detect the images pasted on the drink box to classify the drinks, shown in Figure 11. The red circles on the images are generated by SURF detector.

And as the delivery vehicle, K3 is able to execute the drink identification, autonomous moving, grasping, and place actions, and the tray on K3 is with the maximal capacity of carrying three drinks. With multitasking planning, K3 can treat the multiple tasks as a whole to save the cost for task execution. When a new task is requested, it will be inserted to the current task list, and the individual plan for it will be parallelized into the global plan. As an example, if two persons are coming or leaving at the same time, K3 could serve them in parallel and carry two drinks at once and deliver to them, respectively, shown in Figure 12.

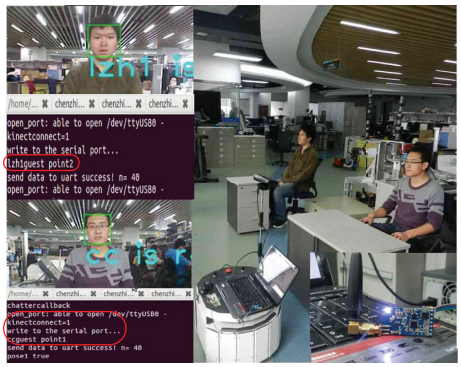


Figure 10: K2 has detected and recognized *Guest lzh1* and *cc* and provided their location.

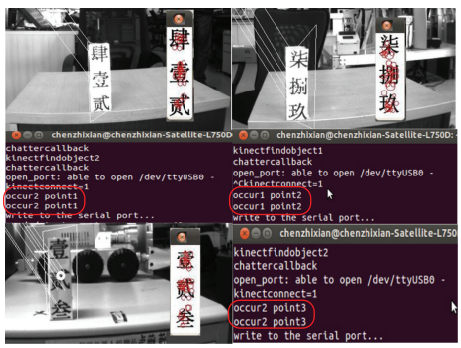


Figure 11: Different drink box at different position is detected precisely.

6.2. Testing and Result. To verify the feasibility and the improvement on the efficiency of the approach proposed, the comparable testing and results are presented in two cases.

Case 1 (serving two guests in parallel by networking). In this case, K3, working with networking, receives a new task that *Guest lzh1* orders *Drink2*, while grasping and placing the *Drink1* on the tray for serving it to *Table1* for *Guest cc*. At this time, the new task is inserted and paralleled into the global plan. As the result of parallelization, two tasks can be executed in parallel, and K3 would bring *Drink1* together with *Drink2* and serve them to *Table1* and *Table2,* respectively. The plan and the executing time are shown in Figure 13, and the waiting time of drink service for *Guest lzh1* is 217 s.

Case 2 (serving two guests by single vehicle). In Case 2, K3 is working as a single vehicle, and it may not be able to detect the coming of new guest in first time, and new service task has to wait until the current task is finished. This takes much more time for the new guest to wait. With the same setting and variables of Case 1, the plan and the executing time are shown in Figure 14, and the waiting time for *Guest lzh1* is 389 s.

As the comparison between Cases 1 and 2, the testing which is based on the proposed approach and networking saves 172 seconds for *Guest lzh1* to get the drink service.

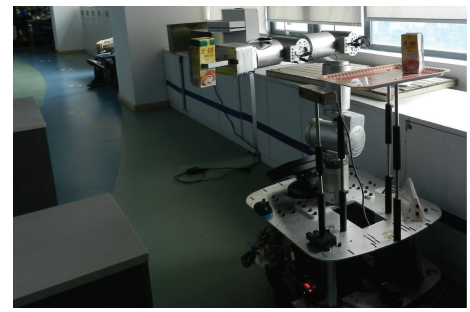


Figure 12: K3 is serving two drinks for two guests in parallel.

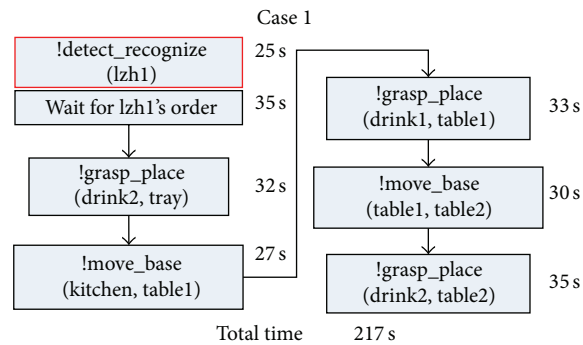


Figure 13: Result of Case 1.

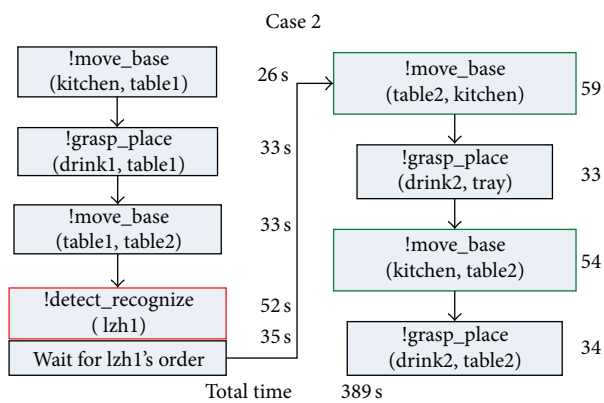


Figure 14: Result of Case 2.

The different bounding colors, red and green, highlight the differences actions between two cases. There, the red color presents the actions which are involved in both cases but with different time costs. In two cases, the time cost for !detect_recognize (lzh1), colored with red, is different. Because in Case 1, K2 can use the shared data of lzh1's face from K1 during executing !detect_recognize (lzh1), while in Case 2, K3 has to obtain this data by itself. At this point, it shows the advance of our method that the networking improves the efficiency on acquiring information. And the green presents the actions that are involved in either of two cases. The actions of !move_base (Table2, Kitchen) and !move_base (Kitchen, Table2) in Case 2 are not appear in Case 1 because K3 has paralleled the two tasks as a global plan. And

it shows the improvement of our approach on the efficiency of multitasking.

7. Conclusion and Future Work

In this paper, we proposed an approach for multitasking in restaurant service of intelligent vehicles by networking. We presented the adapted method of hierarchical decomposing based on traditional HTN planning and the optimization modelled as maximal weight clique problem for optimizing the efficiency of multitasking. And by integrating the plan executing and execution monitoring, the vehicles can work online in dynamic environment. Additionally, as the vehicles are working in network, the mechanism for cooperating by sharing information and allocating task help the vehicles to handle partially observability and uncertainty in dynamic world. As shown in the experiment of restaurant service, three facilities cooperated with each other in a hybrid network and the comparing results between working by networking and working as individual support that the proposed approach has made obvious progress on improving the efficiency and robustness of the vehicles.

Even though the feasibility of the proposed approach is verified, the uncertainty is not really figured out yet. The future works will focus on the deeper integration with the capability for predicting the change of the environment in real world with much more uncertainty, to find the way to endow the indoor intelligent vehicles with more robustness so that it can work more unaffectedly and friendly in a human-robot coexisting world.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is supported by National Natural Science Foundation of China (no. 61210013 and no. 61473278) and Key Research Program of the Chinese Academy of Sciences (KJZD-EW-TZ-L03).

References

[1] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Magazine*, vol. 29, no. 1, pp. 9–19, 2008.

[2] L. Acosta, E. J. González, J. N. Rodríguez et al., "Design and implementation of a service robot for a restaurant," *International Journal of Robotics and Automation*, vol. 21, no. 4, pp. 273–281, 2006.

[3] Q. X. Yu, C. Yuan, Z. Fu, and Y. Z. Zhao, "An autonomous restaurant service robot with high positioning accuracy," *Industrial Robot*, vol. 39, no. 3, pp. 271–281, 2012.

[4] Q.-X. Yu, C. Yuan, Z. Fu, and Y.-Z. Zhao, "Research of the localization of restaurant service robot," *International Journal of Advanced Robotic Systems*, vol. 7, no. 3, pp. 227–238, 2010.

[5] J. López, D. Pérez, E. Paz, and A. Santana, "WatchBot: A building maintenance and surveillance system based on autonomous robots," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1559–1571, 2013.

[6] J. B. Silva, C. Santos, and J. Sequeira, "Developing a timed navigation architecture for hospital delivery robots," in *Proceedings of the IEEE 3rd Portuguese Meeting in Bioengineering (ENBENG '13)*, pp. 1–4, February 2013.

[7] T. Arai, E. Pagello, and L. E. Parker, "Guest editorial—advances in multirobot systems," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 655–661, 2002.

[8] Y. U. Cao, A. S. Fukunaga, and A. B. Kahng, "Cooperative mobile robotics: antecedents and directions," *Autonomous Robots*, vol. 4, no. 1, pp. 7–27, 1997.

[9] A. Torreño, E. Onaindia, and Ó. Sapena, "A flexible coupling approach to multi-agent planning under incomplete information," *Knowledge and Information Systems*, vol. 38, no. 1, pp. 141–178, 2014.

[10] P. H. Tu, T. C. Son, and C. Baral, "Reasoning and planning with sensing actions, incomplete information, and static causal laws using answer set programming," *Theory and Practice of Logic Programming*, vol. 7, no. 4, pp. 377–450, 2007.

[11] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*, Elsevier, 2004.

[12] J. Ma and B.-Q. Yin, "Simulation optimization of actions of robot based on POMDP model," *Journal of System Simulation*, vol. 20, no. 21, pp. 5903–5906, 2008.

[13] S. Candido and S. Hutchinson, "Minimum uncertainty robot path planning using a POMDP approach," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '10)*, pp. 1408–1413, Taipei, Taiwan, October 2010.

[14] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.

[15] M. Fichtner, A. Grossmann, and M. Thielscher, "Intelligent execution monitoring in dynamic environments," *Fundamenta Informaticae*, vol. 57, no. 2–4, pp. 371–392, 2003.

[16] F. R. Noreils and R. G. Chatila, "Plan execution monitoring and control architecture for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 2, pp. 255–266, 1995.

[17] M. E. Desfardins, E. H. Durfee, C. L. Ortiz, and M. J. Wolverton, "A survey of research in distributed, continual planning," *AI Magazine*, vol. 20, no. 4, pp. 13–22, 1999.

[18] J. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, 2nd edition, 2003.

[19] M. Brenner and B. Nebel, "Continual planning and acting in dynamic multiagent environments," *Autonomous Agents and Multi-Agent Systems*, vol. 19, no. 3, pp. 297–331, 2009.

[20] G.-J. M. Kruijff, M. Brenner, and N. Hawes, "Continual planning for cross-modal situated clarification in human-robot interaction," in *Proceedings of the 17th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN '08)*, pp. 592–597, Institute of Electrical and Electronics Engineers Computer Society, Munich, Germany, 2008.

[21] L. Ming, Q. Kejie, C. Fengyu et al., "Towards indoor localization using Visible Light Communication for consumer electronic devices," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '14)*, pp. 143–148, Chicago, Ill, USA, September 2014.

[22] Y. Sun, M. Liu, and M. Q.-H. Meng, "WiFi signal strength-based robot indoor localization," in *Proceedings of the IEEE*

International Conference on Information and Automation (ICIA 14), pp. 250–256, Hailar, China, July 2014.

[23] L. Wang, M. Liu, and M. Q.-H. Meng, "Towards cloud robotic system: a case study of online co-localization for fair resource competence," in *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO '12)*, pp. 2132–2137, December 2012.

[24] R. Micalizio and P. Torasso, "Monitoring the execution of a multi-agent plan: dealing with partial 30 observability," in *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI '08)*, M. Ghallab, C. D. Spyropoulos, N. Fakotakis, and N. M. Avouris, Eds., pp. 408–412, IOS Press, Amsterdam, The Netherlands, 2008.

[25] D. Kortenkamp and R. Simmons, "Robotic systems architectures and programming," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds., pp. 187–206, Springer, Berlin, Germany, 2008.

[26] R. Simmons, S. Singh, D. Hershberger, J. Ramos, and T. Smith, "First results in the coordination of heterogeneous robots for large-scale assembly," in *Experimental Robotics VII*, D. Rus and S. Singh, Eds., vol. 271 of *Lecture Notes in Control and Information Sciences*, pp. 323–332, Springer, Berlin, Germany, 2001.

[27] R. P. Goldman, D. Nau, and U. Kuter, *Documentation for SHOP2*, University of Maryland, 2011.

[28] Q. Yang, "Formalizing planning knowledge for hierarchical planning," *Computational Intelligence*, vol. 6, no. 1, pp. 12–24, 1990.

[29] A. L. Blum and M. L. Furst, "Fast planning through planning graph analysis," *Artificial Intelligence*, vol. 90, no. 1-2, pp. 281–300, 1997.

[30] M. Cayrol, P. Régnier, and V. Vidal, "Least commitment in Graphplan," *Artificial Intelligence*, vol. 130, no. 1, pp. 85–118, 2001.

[31] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo, "The maximum clique problem," in *Handbook of Combinatorial Optimization*, pp. 1–74, Springer, 1999.

[32] P. R. Östergård, "A new algorithm for the maximum-weight clique problem," *Nordic Journal of Computing*, vol. 8, no. 4, pp. 424–436, 2001.

[33] Open Source Robotics Foundation, *Running Ros across Multiple Machines*, Open Source Robotics Foundation, 2009, http://wiki.ros.org/ROS/Tutorials/MultipleMachines.

[34] P. Goebel, Ros by Example Hydro. Lulu, 2013, http://www.lulu.com/shop/r-patrick-goebel/ros-by-example-hydro-volume-1/ebook/product-21393108.html.

[35] P. Ziafati, *A Face Recognition Package for Ros*, GitHub, 2014, https://github.com/procrob/procrob_functional.