Distributed Multi-Robot Task Management

Yikang Gui, Ramviyas Nattanmai Parasurama

***Abstract*—Distributed task assignment for multiple agents raises fundamental and novel problems in control theory and robotics. A new challenge is the development of distributed algorithms that dynamically assign tasks to multiple agents, not relying on a priori assignment information. This work presents a distributed method for multi-robot task management. Our approach handles the conflicts caused by disconnection in distributed multi-robot system by using distance-based and timestamp-based measurement to validate the task allocation for each robot.**

***Index Terms*—Distributed System, Multi-Robot, Task Management**

I. Introduction

SINCE the early 1990s, the problem of task allocation in multi-robot systems has received significant and increasing interest in the research community. As researchers design, build, and use cooperative multi-robot systems, they invariably encounter the question: "which robot should execute which task?" This question must be answered, even for relatively simple multi-robot systems, and the importance of task allocation grows with the complexity, in size and capability, of the system under study. Even in the simplest case of homogeneous robots with fixed, identical roles, intelligent allocation of tasks is required for good system performance, if only to minimize physical interference.

Recent advances in communication and computation have given rise to distributed control of multi-agent systems which, compared to conventional centralized control, provide increased efficiency, performance, and scalability as well as robustness due to its ability to adjust to agent failures or dynamically changing environments. Inspired by these appealing properties of distributed control, we propose a distributed and online solution to the multi-agent dynamic task allocation problem where the assignment of robots to tasks may need to be continuously adjusted depending on changes in the task environment.
In this project, we are aiming to achieve high performance and efficiency in treasure hunting by multiple homogeneous robots. Additionally, we are dealing with distributed multi-robot system instead of centralized multi-robot system. Therefore, we must find a algorithm to deal with the communication between every pair of connected robots.

II. Related Work

Yang, Qin, and Ramviyas Parasuraman. "Hierarchical needs based self-adaptive framework for cooperative multi-robot system." 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2020. In this paper, it introduces a novel SASS framework for cooperation heterogeneous multi-robot systems for dynamic task assignments and automated planning. It combines robot perception, communication, planning, and execution in MRS, which considers individual robot's needs and action plans and emphasizes the complex relationships created through communication between the robots. Specifically, they proposed Robot's Needs Hierarchy to model the robot's motivation and offer a priority queue in a distributed Negotiation-Agreement Mechanism avoiding plan conflicts effectively. Then, they provide several Atomic Operations to decompose the complex tasks into a series of simple sub-tasks. The proposed solution is evaluated through extensive simulations under different static and dynamic task scenarios. The experimental analysis showed that the needs-based cooperation mechanism outperformed state-of-the-art methods in maximizing global team utility and reducing conflicts in planning and negotiation.

Djenadi, Ali, and Boubekeur Mendil. "Energy-aware task allocation strategy for multi robot system." International Journal of Modelling and Simulation (2021): 1-15. This paper presents a distributed energy management task allocation strategy, D-ILS, for multi-robot systems involved in goods transportation. It addresses a task allocation strategy that maximizes productivity and minimizes the energy consumption by considering the energy management in the task allocation. For this purpose, a designed Iterative Local Search (ILS) metaheuristic and a novel heuristic-based utility function are used to solve the multi-robot task allocation problem. In the proposed D-ILS strategy, each individual robot uses the designed ILS to efficiently exploit the search in a local region. The approach takes advantage of the multi-agent aspect of the multi-robot system to better explore the search space.

Fouad, Hassan, and Giovanni Beltrame. "Energy Autonomy for Resource-Constrained Multi Robot Missions." 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2020. In this paper they present a control barrier function (CBF) based framework for long term autonomy of multi robot systems with limited charging resources. They started by highlighting some tracking properties of the energy persistence CBF and then they introduced a CBF based framework to achieve the necessary coordination for sharing the charging station. They investigate the system capacity in terms of the relation between feasible requirements of charging cycles and individual robot properties. They show simulation results, using a physics-based simulator and real robot experiments to demonstrate the effectiveness of the proposed approach.

III. Methodology

A. Robot Setting

In order to achieve completely isolated robots, we define the class *RobotStatusManager* in Python. Each initialization of class *RobotStatusManager* represents a robot.

B. Connectivity and Disconnectivity

The connectivity and disconnectivity represents the status of a robot that whether it is connected or disconnected to other robots, respectively. Every pair of robots should have either connectivity or disconnectivity.

C. Task Receipt

A task receipt is a message contains status information of a robot. The content includes:

- Task type: Currently, there are 7 task types including IDLE, GO_TO_PICK, GO_TO_COLLECTION, GO_TO_RECHARGE, RECHARGING, WAIT_FOR_RECHARGING, DEAD.
- Task location: The coordinates of current task
- Robot id: The robot id for this message
- Previous task location: The coordinates of previous one task, only useful when current task is GO_TO_COLLECTION
- Timestamp: Current global timestamp
- Priority: Useful for validation of this message
- Distance to task location: The Euclidean distance to current task

D. Robot Task Schedule

In the *RobotStatusManager*, there is a *Task Controller* to control the task schedule stored in every robot. The task schedule stores the information about itself and all other robots. The information about other robots may be outdated due to the disconnectivity

E. Robot Task Controller

A *TaskController* is used for managing all task receipt of one robot. The number of task receipts should be the same as the number of robots. A fully functional *TaskController* should return the status of every tasks in the environment for future purpose, such as task assignments. In our case, the function is named by *get_sim_status*.

F. Simulator

A *simulator* is the platform where the simulation is executed. The simulator is isolated from the function of robots, which means the simulator cannot directly manipulate the robots but it can call the robots' *set* and *get* function defined in the robot class to simulate the movement of the robot.

The simulator includes the following function:

- Check the connectivity of each pair of robots and calculate the distance between the robots and their target locations.
- Check the validation of the task schedule in each robot by timestamp and distance.
- Check the expiration of the task schedule in each robot and assign IDLE task to those expired task.
- Call Robotarium functions to process the simulation and make a movement.
- Other auxiliary function, such as visualization and statistics.

1) Check Connectivity: This is the first function provided by the simulator. Since in our setting, every robot has the same sensing range. The robot can communicate with its neighbors only when its neighbors are in its sensing range. It is very important for the robot to determine whether their neighbors are connected or not. In order to achieve this, we define a algorithm named by Check Connectivity. The procedure is illustrated in Algorithm 1.

Algorithm 1 Check Connectivity

```
1: Calculate the distance between each pair of robots
2: Keep the pair of robots whose distance between them is
   smaller than the sensing range
3: Save the above pair of robots for future purpose
```

2) Check Expiration: This is the second function provided by the simulator. In order to avoid the scenario that one robot was broadcast with all other robots in the beginning and it got disconnected in the rest simulation. In that case, the robot will never take a movement instead of waiting for reconnected forever. Therefore, we introduce a mechanism called *expiration*. The task receipts saved in each robot task schedule will have a expiration. Once the task receipts is expired, the robot task schedule will assign an idle task receipt to targeted robot. Here is a difference in the task receipt assigned after expiration that the priority will assign to 0 instead of 1. Priority 1 means that this task receipt is assigned by its real owner. Priority 0 means that the task receipt is assigned after an expiration and it is not assigned by its real owner. Here the owner means that the robot with the *robotID* in the task receipt.

Here is an example. Assume that there are two robots, robot A and robot B, in our simulation. The expiration threshold is 400 and there is only one treasure location. Robot A and robot B are initialized close enough to make sure they are connected. Robot A is assigned to take the treasure and Robot B is assigned idle at timestamp 0. Due to the connectivity, robot A and robot B have the same task schedule, in which robot A is recorded to take the treasure and robot B is recorded to be idle. Then robot A goes to fetch the treasure and robot A and robot B get disconnected in the rest simulation. At timestamp 401, robot B will update the status of robot A in its task schedule with idle due to the expiration of robot A's task receipt. what's more, since the task receipt of robot A in robot B's *TaskController* is updated by robot B, the priority of this task receipt is 0. Algorithm 2 depicts the procedure to fix the expiration.

Algorithm 2 Check Expiration

```
1: EXPIRATION is the threshold for expiration
2: for Every TaskReceipt in task schedule do
3:     if TaskReceipt.timestamp - current_timestamp > EXPIRATION then
4:         if TaskReceipt.priority > 0 then
5:             if TaskReceipt.robotID != its own robotID then
6:                 TaskReceipt = IDLE TaskReceipt with priority = 0
7:             end if
8:         end if
9:     end if
10: end for
11: return task schedule
```

3) Check Validation: This is the third function provided by the simulator. This function is designed to handle the conflicts caused by the disconnectivity of each pair of robots.

The first set of conflicts is caused by the setting of environment, the conflicts happen when two or more robots are assigned the same treasure and they are to go to pick the treasure. In this case, the robot closest to the treasure will keep the task and the rest robots will re-assign an idle task.

The second set of conflicts is all the rest possible conflicts. We can use timestamp and priority to handle this set of conflicts. When conflicts happen, we first check the priority of both task receipts. The task receipt with lower priority will be replaced by the task receipt with higher priority. If two task receipts have the same priority, the task receipts with the latest timestamp with replace the other task receipt. Table I demonstrates the measurements to handle the conflicts. Algorithm 3 conveys the procedure to handle the conflicts.

TABLE I: Measurement of Validation

Robot A Action	Robot B Action	Measurement
Pick	Pick	Distance
Pick	Collection	Timestamp and priority
Collection	Collection	Timestamp and priority
Pick	Idle	Timestamp and priority
...	...	Timestamp and priority

IV. Experiments

A. Setting

- N=5 number of robots.
- T=5 fixed treasure locations.
- R=2 fixed recharger stations at top right and top left of the workspace.
- C=1 one fixed collection point at bottom left of the workspace.
- I=10000 iterations for simulation
- Starting position of the robots = random within the workspace.
- Distance clearance threshold = 0.1m (to decide if a robot has reached a location or not).
- All distances are expressed in Euclidean norm and time is expressed as iterations.

Algorithm 3 Check Validation

```
1: for TaskReceiptA, TaskReceiptB in TaskScheduleA, TaskScheduleB do
2:     if TaskReceiptA.priority > TaskReceiptB.priority then
3:         TaskReceiptB = TaskReceiptA
4:     else if TaskReceiptA.priority < TaskReceiptB.priority then
5:         TaskReceiptA = TaskReceiptB
6:     else if TaskReceiptA.timestamp > TaskReceiptB.timestamp then
7:     else if TaskReceiptA.timestamp < TaskReceiptB.timestamp then
8:         TaskReceiptA = TaskReceiptB
9:     end if
10: end for
11: while Multiple TaskReceipts have PICK task and have same treasure
12:  do keep the robot closest to the treasure and reassign IDLE to rest robots
13: end while
```

- Disable the collision avoidance and barrier function in Robotarium.

Here is an example of the environment. The line between two robots represents they are connected. The black dots represent treasure, recharge station and collection point.

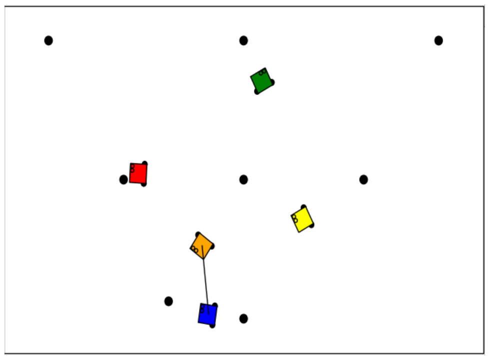

Fig. 1: Environment for experiments

Assumption: Every task location has unlimited amount of treasure available. When a robot is at the task location, then it is assumed that the robot picked up the treasure (if it does not have one already). When a robot reached a collection point, then it is assumed that it dropped the treasure (if it already had one with it).

Definition of Task completion: A task is said to be complete when a robot reaches any of the task location, and it had then dropped that treasure at the collection point.

Energy model: A robot will lose energy based on the following energy consumption model.

$$E(t) = E(t-1) - (\alpha * 1 + \beta * distance + \gamma * IsTreasure)$$

A robot will gain energy if it is at the recharging station with the following model:

$$E(t) = E(t-1) + \delta * 1$$

Use the following values:

$$\alpha = 0.01, \beta = 0.1, \gamma = 0.02, \delta = 0.1$$

. A robot will lose energy based on the following energy consumption model:

$$E(t) = E(t-1) - (\alpha * 1 + \beta * distance + \gamma * IsTreasure)$$

A robot will gain energy if it is at the recharging station with the following model:

$$E(t) = E(t-1) + \delta * 1$$

Use the following values:

$$\alpha = 0.01, \beta = 0.1, \gamma = 0.02, \delta = 0.1$$

B. Result

1) Experiment 1: N=5, T=5, R=2, C=1 I=10000, enable expiration, sensing range = 0.5, 20 experiments

TABLE II: Experiment 1

	Mean	Std
Travel Distance	223.71	2.41
Go to recharge time	5009.15	157.74
Wait for recharge time	1785.4	360.20
Recharge time	8517.9	245.76
Treasure completion	97.55	1.74

2) Experiment 2: N=5, T=5, R=2, C=1 I=10000, enable expiration, sensing range = 5, 20 experiments

TABLE III: Experiment 2

	Mean	Std
Travel Distance	227.35	2.36
Go to recharge time	5019.25	174.24
Wait for recharge time	1848.2	343.65
Recharge time	8606.0	186.35
Treasure completion	95.65	1.71

3) Experiment 3: N=5, T=5, R=2, C=1 I=10000, disable expiration, sensing range = 0.5, 20 experiments

TABLE IV: Experiment 3

	Mean	Std
Travel Distance	225.96	2.44
Go to recharge time	4969.875	176.16
Wait for recharge time	1825.3	300.58
Recharge time	8502.45	163.33
Treasure completion	96.42	1.57

4) Experiment 4: N=5, T=5, R=2, C=1 I=10000, disable expiration, sensing range = 5, 20 experiments

TABLE V: Experiment 4

	Mean	Std
Travel Distance	227.07	2.05
Go to recharge time	5061.1	163.69
Wait for recharge time	1820.55	206.96
Recharge time	8567.35	159.22
Treasure completion	95.75	1.54

5) Experiment 5: N=5, T=2, R=2, C=1 I=10000, enable expiration, sensing range = 0.5, 20 experiments

TABLE VI: Experiment 5

	Mean	Std
Travel Distance	197.40	3.83
Go to recharge time	4482.45	125.65
Wait for recharge time	1420.9	310.76
Recharge time	8048.4	188.27
Treasure completion	78.0	1.67

TABLE VII: Experiment 6

	Mean	Std
Travel Distance	182.44	8.77
Go to recharge time	4642.31	173.24
Wait for recharge time	1476.73	395.65
Recharge time	7979.42	210.46
Treasure completion	71.42	3.67

6) Experiment 6: N=5, T=2, R=2, C=1 I=10000, disable expiration, sensing range = 0.5, 20 experiments

V. Discussion and Conclusion

The expiration mechanism can significantly increase the performance when the number of treasure is smaller than the number of the robot. What's more, the sensing range is another important factor. The larger sensing range, the closer performance compared to centralized manner.

The future improvement includes adding the collision avoidance and barrier function in Robotarium. And transfer the energy management in centralized method to distributed method.

References

[1] Yang, Qin, and Ramviyas Parasuraman. "Hierarchical needs based self-adaptive framework for cooperative multi-robot system." 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2020.

[2] Djenadi, Ali, and Boubekeur Mendil. "Energy-aware task allocation strategy for multi robot system." International Journal of Modelling and Simulation (2021): 1-15.

[3] Fouad, Hassan, and Giovanni Beltrame. "Energy Autonomy for Resource-Constrained Multi Robot Missions." 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2020.

[4] N. Michael, M. M. Zavlanos, V. Kumar and G. J. Pappas, "Distributed multi-robot task assignment and formation control," 2008 IEEE International Conference on Robotics and Automation, 2008, pp. 128-133, doi: 10.1109/ROBOT.2008.4543197.

[5] Firoozi, Roya, et al. "A Distributed Multi-Robot Coordination Algorithm for Navigation in Tight Environments." arXiv preprint arXiv:2006.11492 (2020).