De-centralized Resource-based Multi-Robot Task Allocation

CSCI: 8535 – Multi-Robot Systems

Shailendra Sekhar Bathula
Institute of Artificial Intelligence
University of Georgia
Athens, GA.

ABSTRACT: *In this study, we aim to comprehensively investigate and implement a decentralized robotic system that considers individual resource constraints prior to task assignment. Our objective is to develop an optimal task allocation mechanism, leading to enhanced overall system efficiency and performance.*

1. Introduction

Task allocation is one of the core problems of multi-robot systems. The ability to assign the right tasks to the right robots enables robotic enthusiasts to manipulate robots to perform tasks cooperatively and complete them in the least possible time. Predominant task allocation algorithms are either based on utility or combinatorial optimization. Utility-based algorithms, also known as auction algorithms, provide the flexibility to build systems irrespective of their sensor types. Each robot considers its own parameters and comes up with a utility value also referred to as bids for each task available. Based on the objective of the task, either the robot with the minimum or maximum bid wins the task they are bidding for. Auction-based algorithms also offer very high explain ability and scalability. Auction-based algorithms can be implemented in both centralized and decentralized ways. Centralized systems are where every robot has access to all the information at any given point in time. Centralized systems are effectively like building a common brain where all the calculations are done for all the robots. If there is any communication delay or outage the entire system breaks down. De-centralized approaches offer a solution to this problem. Here every robot has its own brain. It individually decides what it must do or where it must go. Any form of communication delay or outage will only result in the expulsion of a particular robot from the system.

Multi-robot task allocation classifies a problem into a combination of single-task robot or multi-task robot, single robot task or multi-robot task, and instantaneous assignment or time extended assignment. Single-task robots mean each robot is capable of one task at a time while multi-task robots can execute multiple tasks. Single-robot task implies each task requires one robot to achieve it while multi-robot tasks are those that require more than one robot. An instantaneous assignment is when the task allocation is done at every instance without taking its future situation into consideration. The time-extended assignment has more information available to allocate tasks and has the capability to allocate asynchronously.

Task allocation by itself is a difficult problem, but useless in real-world scenarios if the resource of each robot is not taken into consideration. Only by taking resources into consideration, the system can be successful in its operations. Therefore, these systems should know how to exploit and recharge their resources while they are performing tasks.

This paper talks about an attempt to build a decentralized auction-based task allocation system that takes resources into consideration. The project is to build a decentralized system for robots to complete a set of tasks. The algorithm this paper talks about can be classified as a single task robot – single robot tasks – time extended assignment.

2. SYSTEM ARCHITECTURE

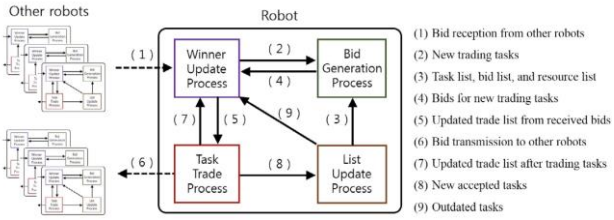

Fig1: System architecture

This system we are trying to build is basically the brain of each robot. We have 4 different processes happening inside each robot. The robot receives a task in the winner update process, then it requests a bid generator to calculate the cost for this task. The bid generator in turn uses the information from the list update process where information regarding the robot is stored. Now, the winner update process sends information about the task and its bid to the task trade process. The task trade process is where the robot decides to accept or reject a task. If accepted, the information about the accepted task is stored in the list update process.

2.1 Winner Update Process

The winner update process deals with 2 lists. A bid list is a list of lists containing information about various tasks the robot needs to compute for. Each list in the bid list is called a bid vector which contains information about the task in question, the robot to which it was initially allocated or the robot who sent it, the bid of that robot for the task, and if the message is from an auctioneer or not. A trade list is a list of lists where each robot maintains information about the best candidate for a particular task with its bid information, the priority of the task for the candidate, and if the best candidate is the auctioneer or not. Each list in the trade list is a trade vector. The winner's update process starts by evaluating each bid vector. First, it checks if the vector is from an auctioneer or not and if we have any prior information about this task in our trade list. The system is designed to process information only if the message is from an auctioneer and we don't have any prior information about

the task or if we have some information about this task in our trade list.

Algorithm 2 Winner update process of robot i

procedure *WinnerUpdate*($\mathbf{Q}^i$, $\mathbf{Z}^i$)
  for $n = 1$ to $|\mathbf{Z}^i|$ **do**
    $j = z^i_{n,1},\ k = z^i_{n,2},\ b^k_j = z^i_{n,3},\ auc = z^i_{n,4}$
    if $auc =$ **True** $\wedge\ Q^i_j \notin \mathbf{Q}^i$ **then**
      $[\hat{b}^i_j\ \hat{\mathbf{x}}^i_j\ n_j] = BidGen(j)$
      if $i = k$ **then**
        $Q^i_j = [i\ \hat{b}^i_j\ \hat{\mathbf{x}}^i_j\ n_j$ **True**$]$
      else if $\hat{b}^i_j < b^k_j$ **then**
        $Q^i_j = [i\ \hat{b}^i_j\ \hat{\mathbf{x}}^i_j\ n_j$ **False**$]$
      else
        $Q^i_j = [k\ b^k_j\ \vec{0}\ 0$ **False**$]$
      end if
      $\mathbf{Q}^i.PushBack(Q^i_j)$
    else if $Q^i_j \in \mathbf{Q}^i\ \wedge\ b^k_j < q^i_{j,2}$ **then**
      $Q^i_j = [k\ b^k_j\ \vec{0}\ 0\ q^i_{j,5}]$
    end if
  end for
  $\mathbf{Z}^i = \emptyset$
  return $\mathbf{Q}^i$
end procedure

Let us consider the case where the message is indeed from the auctioneer, and we don't have any information about the task in our trade list. The robot immediately requests the bid generator process to generate a suitable bid, resource levels after the task, and the suitable priority for the task. If the message is from the robot itself, generally when this robot is the first to detect a task, then it generates a trade vector with its own information and stores it in its trade list. If the message is not from the robot itself, but our newly calculated bid is lower than the bid of the task proposed, the robot generates a trade vector with its information but sets the auctioneer to *False*. Otherwise, the robot simply generates a trade vector with information received from the bid list itself. If the robot already has some information about the task in its trade list, it checks if the new bid calculated is less than the one in its trade vector. If it is, then it updates the trade vector with its newly calculated values.
This process ends when all the bid vectors are processed.

2.2 Bid Generation

Let us take a step back and understand the working of bid generation we used in the winner update process. The bid generator process is designed in a way that the bids generated by the process take the resources into consideration. The expected outputs from the bid generator process are the bids for the task, its priority for the task, and the resources the robot would be left with after the completion of the task. To calculate the resource robot would be left with after the completion of a task, the robot must know about the tasks it has to execute, the resources required for those tasks, the current resource level, and the threshold for the new task. The threshold for a given task is calculated by the distance between the new task and the nearest charging station. This kind of threshold helps robots maintain their resources when executing a task.

Algorithm 1 Bid generation process of robot i for task j

procedure *BidGen*(j)
  for $n'_j = 1$ to $\min(|\mathbf{A}^i|+1, N_A)$ **do**
    $\hat{c}^i_j = \sum_{k=1}^{N_P} P^i_j(k)\ c^i_j(A^i_{n'_j-1}, k)$
    if $\hat{c}^i_j < C^i_{n'_j}\ \vee\ n'_j > |\mathbf{A}^i|$ **then**
      $n_j = n'_j$
      if *Obj=MiniSum* **then**
        $\hat{b}^i_j = \hat{c}^i_j$
      else if *Obj=MiniMax* **then**
        $\hat{b}^i_j = B^i_{n_j-1} + \hat{c}^i_j$
      end if
      $\hat{\mathbf{x}}^i_j = \sum_{k=1}^{N_P} P^i_j(k)\ \mathbf{x}^i_j(A^i_{n_j-1}, X^i_{n_j-1}, k)$
      break
    end if
  end for
  return $\hat{b}^i_j,\ \hat{\mathbf{x}}^i_j,\ n_j$
end procedure

In this project, a bid is defined as the distance a robot must travel to execute a specific task. Resource depletion is directly proportional to the distance traveled by the robot. To evaluate the robot's remaining resource levels, simply subtract the resources consumed at previous task locations or the robot's current location from the total resources available. We first assess whether the remaining resource levels are greater than the predetermined threshold. If they are, we record the distance as the bid and the resources remaining after task completion. If not, the bid for the task will be equivalent to the sum of the distance between the robot and the charging station, and the distance from the charging station to the task location. In this scenario, the resources after task completion will be equivalent to the maximum charge the robot acquires at the charging station, minus the distance between the charging station and the task. We assign a priority of 0 in this case, as no other tasks have been allocated. When a robot already has tasks assigned, it calculates the distance between each allocated task and the new task, estimates the resource level after executing the new task, and computes the bid in a manner similar to the previous method. If the newly calculated bid is lower than the bid of a task in the allocated list, we consider that position in the list as a potential suggestion for the robot to execute the new task, if necessary.

2.3 Task Trade Process

Now coming back to our main story of trade lists where information about the best candidates for the tasks is stored.
The task trade process has two primary objectives. The first objective is to identify the most suitable task from the newly introduced trading tasks for the robot and subsequently transmit a bid vector to its neighboring robots' bid lists, prompting them to submit their respective bid values. The second objective, when a robot serves as the auctioneer for a task, is to evaluate which robot should be awarded the task and communicate the winning bid for acceptance or rejection.
To determine the optimal task, a robot examines its trade list and identifies the task with the lowest bid as the best option. Once the robot selects the most suitable task, it generates a

bid vector for the task and transmits it to its neighboring robots. If no response is received within a specified time frame, the auctioneer robot will self-allocate the task. Upon completion of this process, the robot removes the trade vector from its trade list.

Algorithm 3 Task trade process of robot i

procedure $TaskTrade(\mathbf{Q}^i)$
  $Q_m^i = \vec{0}$
  if $j^\star = 0$ **then**
    $j^\star = \arg\min_{j \in \{j | Q_j^i \in \mathbf{Q}^i, q_{j,1}^i = i\}} q_{j,2}^i$
    $Z_{tmp} = [j^\star \; i \; q_{j^\star,2}^i \; q_{j^\star,5}^i]$
    $Transmit(Z_{tmp})$
    $\mathbf{Q}^i = [Q_{j^\star}^i]$
  else if $t_{wait} > t_{delay}$ **then**
    if $q_{j^\star,1}^i = i$ **then**
      $Q_m^i = Q_{j^\star}^i$
    end if
    if $q_{j^\star,5}^i =$ **True then**
      $Z_{tmp} = [j^\star \; q_{j^\star,1}^i \; q_{j^\star,2}^i$ **False**$]$
      $Transmit(Z_{tmp})$
    end if
    $\mathbf{Q}^i.Remove(Q_{j^\star}^i), \; j^\star = 0$
  end if
  return $\mathbf{Q}^i, \; Q_m^i$
end procedure

Algorithm 4 List update process of robot i

procedure $ListUpdate(\mathbf{Z}^i, Q_m^i, R_S)$
  if $Q_m^i \neq \vec{0}$ **then**
    $n_m = q_{m,4}^i$
    $Z_a^i = [j \; i \; b_D$ **True**$], \; \forall j \in \mathbf{A}_{n_m:|\mathbf{A}^i|}^i$
    $\mathbf{Z}^i.PushBack(Z_a^i)$
    $\mathbf{A}^i = \mathbf{A}_{1:n_m-1}^i, \; \mathbf{B}^i = \mathbf{B}_{1:n_m-1}^i, \; \mathbf{X}^i = \mathbf{X}_{1:n_m-1}^i$
    $\mathbf{A}^i.PushBack(m), \; \mathbf{B}^i.PushBack(q_{m,2}^i), \; \mathbf{X}^i.PushBack(q_{m,3}^i)$
  end if
  if $R_S = Perform$ **then**
    if $J = 0$ **then**
      $J = A_1^i$
      $k^\star = \arg\max_k P_j^i(k), k \in \{1, 2, ..., N_P\}$
    else if J is completed **then**
      if *Obj=MiniMax* **then**
        $B_n^i = B_n^i - B_1^i, \; \forall n \in \{1, 2, ..., |\mathbf{A}^i|\}$
      end if
      $\mathbf{A}^i.PopFront(), \; \mathbf{B}^i.PopFront(), \; \mathbf{X}^i.PopFront()$
      $J = 0$
    end if
  else if $R_S = Refill \; \wedge \; J \neq 0$ **then**
    $J = 0$
    $Z_a^i = [j \; i \; b_D$ **True**$], \; \forall j \in \mathbf{A}^i$
    $\mathbf{Z}^i.PushBack(Z_a^i)$
    $\mathbf{A}^i = [\,], \; \mathbf{B}^i = [\,], \; \mathbf{X}^i = [\,]$
  end if
  return $\mathbf{A}^i, \; \mathbf{B}^i, \; \mathbf{X}^i, \; \mathbf{Z}^i$
end procedure

2.4 List Update Process

The list update process maintains information about the tasks a robot needs to perform, bid values, and resources after each task. When the robot gets an allocation of a task, information from the input trade vector is updated into the robot's lists. The list update process is also responsible for determining the status of the robot and manage lists accordingly. If the robot has resources higher than thresholds and tasks in the allocated list, it sets the first task in the list for the controller to execute. Once the task is completed, information from this task is removed from its lists. If the robot's resource levels are below the threshold, then the robot creates bid vectors for all the tasks in its allocated list and sends them back to the bid list in the winner update process. Ultimately, it clears all its lists and heads toward the charging station.

3. Discussion

The task allocation and control system presented in this report offers several key advantages. Firstly, the system optimizes task allocation based on the distance to tasks and the battery levels of the robots. This ensures that tasks are assigned to robots with sufficient battery capacity, minimizing the risk of task abandonment due to depleted batteries. Secondly, the system incorporates a positional controller to regulate the velocities of the robots, allowing them to move efficiently toward their allocated tasks. Thirdly, the system integrates battery management, enabling robots to recharge at the designated battery station when their battery levels fall below a threshold. This feature prolongs the overall mission time and ensures continuous task execution. The system's consideration of battery levels addresses a critical challenge in multi-robot systems. Energy limitations can significantly impact the operational duration and effectiveness of robots. By dynamically allocating tasks based on battery levels and enabling robots to recharge when needed, the system maximizes the robots' productive time and minimizes downtime due to battery depletion. This feature increases overall system efficiency and ensures continuous task execution. While the system showcased promising results, there are several areas for further improvement and exploration. The system's control strategies could be enhanced to incorporate collision avoidance techniques and path-planning algorithms. These improvements would enable smoother and more efficient robot movements, ensuring safety and avoiding potential collisions in dynamic environments.

4. Conclusion

In conclusion, the task allocation and control system presented in this paper offers a comprehensive solution for efficient task allocation and resource utilization in multi-robot environments. By considering resource constraints, specifically battery levels, the system optimizes task allocation, minimizes travel distances, and ensures uninterrupted task execution. The system holds significant potential for real-world applications in domains such as robotics, logistics, and automation, where multi-robot systems play a vital role in achieving complex objectives.

5. References

1. D.-H. Lee, Resource-based task allocation for multi-robot systems, Robotics and Autonomous Systems (2018), https://doi.org/10.1016/j.robot.2018.02.016
2. Gerkey BP, Matarić MJ. A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. The International Journal of Robotics Research. 2004;23(9):939-954. doi:10.1177/0278364904045564