

We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

5,600

Open access books available

137,000

International authors and editors

170M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



A Distributed Approach for Autonomous Cooperative Transportation

Amar Nath and Rajdeep Niyogi

Abstract

Autonomous mobile robots have now emerged as a means of transportation in several applications, such as warehouse, factory, space, and deep-sea where direct human intervention is impossible or impractical. Since explicit communication provides a better and reliable way of multi-robot coordination compared to implicit communication, so it is preferred in critical missions, such as search and rescue, where efficient and continuous coordination between robots is required. Cooperative object transportation is needed when the object is either heavy or too large or needs extra care to handle (e.g., shifting a glass table) or has a complex shape, which makes it difficult for a single robot to transport. All group members need no participation in the physical act of transport; cooperation can still be achieved when some robots transport the object, and others are involved in, say, coordination and navigation along the desired trajectory and/or clear obstacles along the path. A distributed approach for autonomous cooperative transportation in a dynamic multi-robot environment is discussed.

Keywords: cooperative transportation, distributed algorithm, dynamic environment, multi-agent coordination and cooperation

1. Introduction

Autonomous mobile robots are now used in several applications, such as warehouse, factory, space, and deep-sea, that may be inaccessible for humans. The main concern is to find an effective coordination mechanism among autonomous agents to perform tasks in order to achieve high quality overall performance. Although MAS research has received substantial attention, multi-robot coordination remains a challenging problem since the overall performance of the system is directly affected by the quality of coordination and control among the robots while executing cooperative tasks. Coordination in a multi-robot system can be achieved either by explicit or by implicit communication. Since explicit communication provides a better and reliable way of multi-robot coordination compared to implicit communication, so it is preferred in critical missions, such as search and rescue, where efficient and continuous coordination between robots is required.

A collaborative task cannot be executed by any single agent. It requires multiple agents at the task's location. Execution of such tasks is quite challenging in a dynamic environment, as the time and location of a task arrival, required skills, and the number of robots required for its execution may not be known a priori. This necessitates the design of a distributed algorithm for collaborative task execution

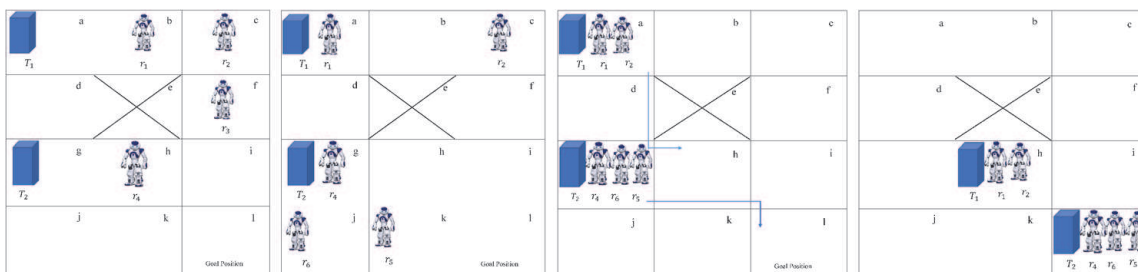


Figure 1.
Snapshots of a dynamic environment.

via runtime team/coalition formation. To form a team with a lack of global knowledge, the robots need to communicate with each other to acquire relevant information. Here, a distributed approach for collaborative task execution in a dynamic environment is discussed. We illustrate the applicability of the approach with urban search and rescue (USAR) domain and evaluate its performance with extensive experiments using ARGoS, a realistic multi-robot simulator.

We now illustrate an example scenario of the problem considered as shown in **Figure 1**. The environment, a grid world of size 4×3 , consists of 12 locations marked as a, b, \dots, l . Robots can move to its adjacent location. Boxes arrive at the locations at different points in time. The task is to move a box from its arrival location to the goal location, which is marked on the box.

The snapshots of the environment at different instants of time is shown in **Figure 1**. At time t_1 , two tasks τ_1 and τ_2 arrive at locations a and g respectively, 4 robots are present at the locations b, c, f and h . The robots r_1 and r_4 detect the tasks τ_1 and τ_2 respectively. At time t_2 , r_1 and r_4 move to locations a and g to attend the tasks. Now, r_1 and r_4 determine the team sizes to be 2 and 3, and the goal locations to be h and l for the tasks τ_1 and τ_2 respectively. At this time, r_3 exits and r_5 and r_6 enter at locations k and j respectively.

At t_2 , r_1 and r_4 do not know the states and locations of other robots present in the environment, and thus with this insufficient information they cannot form their respective teams. Thus, in order to form their teams, they invoke the algorithm given in Section 4. At t_3 , r_1 and r_4 both form their teams successfully and the members reach the locations of the tasks as shown in the Figure. Finally, at time t_4 , execution of the tasks are completed and the team members for τ_1 and τ_2 reach their respective goal locations.

2. Related work

In the literature, several approaches have been suggested for solving the problem of cooperative object transportation [1–4]. The work [1] is considered as the pioneering work, targeting a cooperative transport task by a homogeneous group of simple robots that can only push an object. The authors [1] demonstrate that coordinated effort is not possible without explicit communication.

The work [2] proposed direct (explicit) communication to improve the coordination of a homogeneous group of two six-legged robots required to transport a rectangular box towards a target cooperatively. The work [3] considered the problem of cooperative box pushing where the roles of the members are pre-defined; specifically one robot acts as a watcher and the others act as pusher. However, we consider a more complex scenario of cooperative object transportation scenario, where the role of each robot is not fixed in advance, rather decided at runtime. In [4], the robots are designed to push the object across the portion of its surface, where it occludes the direct line of sight to the goal. This simple behavior results in transporting the object towards the goal without using any form of direct communication.

The problem of cooperative transportation, considered in this paper, involves team formation of heterogeneous robots and gathering the robots to the location of the object to be transported. The number of robots required to transport the object is not known a priori and it is decided at runtime. For the same task, the team size is determined by the state of the environment. So, at some point in time an object may be transported by two robots while at some other moment in time three or more robots are required. Few works related to coalition formation strategies are discussed below.

Auction-based approaches for team formation (task allocation) are suggested in [3, 5]. A bidder agent has some resources (e.g., data center, CPU) [5], who may bid for multiple auctioneers concurrently. However, when we move to physical agents, a robot cannot be a member of multiple coalitions at any point of time simply because the tasks may be at different locations, and a robot cannot be at two different locations at the same time, even though a robot may have the capability to perform multiple tasks at a time.

In our work, a non-initiator robot (bidder) will not express its willingness to multiple initiators (auctioneers) concurrently; when more than one request message arrives, the robot stores the requests in its local queue. Having one or more resources specified in the auction is a sufficient condition for an agent to make a bid [5]. Having the required skills for a task is a necessary but not a sufficient condition for a robot to express its willingness to be part of a team, in our work. A robot's behavior, in our work, is determined by its current state, whereas in [3, 5] states need not be taken into consideration.

In [6], the authors describe a framework for dynamic heterogeneous team formation for robotic urban search and rescue. The task discovery is made by a member of a team and it is sent to the team coordinator for assignment. The team coordinator performs the task assignment ensuring the task will be carried out by a robot with the necessary capabilities. However, in a distributed system, no robot knows the states, locations, and skills of other robots. Thus, the robots should communicate among themselves to acquire relevant information for task execution without the intervention of any central authority. This necessitates the design of a distributed algorithm for task execution in such a dynamic environment.

In our approach, unlike [6], every robot has a similar level of priority, and each of them can perform the task management activities, i.e., searching, team/coalition formation by acquiring the information from the robots available in the environment at that moment in time. In this paper, the arrival time and location of a task are not known a priori; hence, task searching and coalition formation activities are performed by a robot at runtime.

3. Problem formalization

A formal framework of a dynamic environment and some related concepts are presented below.

Definition 3.1. (Dynamic environment) A global view (snapshot) of an environment \mathcal{E} , with a set of locations L , taken at time t , is given by a 3-tuple $\mathcal{E}^t = \langle \mathcal{R}^t, \mathcal{T}^t, f \rangle$, where \mathcal{R}^t is the set of robots present in the environment at time t , and \mathcal{T}^t is the set of tasks that arrive in the environment at time t , $f : \mathcal{R}^t \times \mathbb{N} \mapsto L$, is a function that gives the location of a robot at a discrete instant of time represented by the set of natural numbers \mathbb{N} .

A robot has a set of skills ψ (eg., gripper, camera), and at any instant of time it may be in any state from the set of states $\mathcal{S} = \{Idle, Ready, Promise, Busy\}$. A robot can enter the environment \mathcal{E} at any time, but can leave only if its state is *Idle*. When

a robot attends a task, it can determine the information required to begin team formation, from the task specification, which is given below.

Definition 3.2. (Task (cooperative transportation)) A task τ is specified by a 5-tuple $\tau = \langle \nu, l, t, k, \Psi \rangle$ where ν is the name of a task (e.g., move (carry) box B to location l' , lift desk D), $l \in L$ is the location where the task arrived, t is the time at which the task arrived, $k > 1$ is the number of robots required to execute the task, and Ψ is the set of skills required to execute the task.

Definition 3.3. (Condition for single task execution) A task $\tau = \langle \nu, l, t, k, \Psi \rangle$ can be executed, if there exists a set \mathcal{R} of k available robots, such that for all $r \in \mathcal{R}$, $\psi_r \supseteq \Psi$ at some time $t' > t$, and for all $r \in \mathcal{R}$, location of r , $loc_r = l$ at some time $t'' > t'$.

The first condition in the *if* is for team formation, and the second condition is for ensuring that all the team members converge to the location of the task.

Definition 3.4. (Condition for multiple task execution) The tasks $\tau_1 = \langle \nu_1, l_1, t, k_1, \Psi_1 \rangle$ and $\tau_2 = \langle \nu_2, l_2, t, k_2, \Psi_2 \rangle$ can be executed if the following conditions hold:

1. there exists a set \mathcal{R}_1 of k_1 available robots, such that for all $r \in \mathcal{R}_1$, $\psi_r \supseteq \Psi_1$ at some time $t'_1 > t$, and for all $r \in \mathcal{R}_1$, $loc_r = l_1$ at some time $t''_1 > t'_1$.
2. there exists a set \mathcal{R}_2 of k_2 available robots, such that for all $r \in \mathcal{R}_2$, $\psi_r \supseteq \Psi_2$ at some time $t'_2 > t$, and for all $r \in \mathcal{R}_2$, $loc_r = l_2$ at some time $t''_2 > t'_2$.
3. $\mathcal{R}_1 \cap \mathcal{R}_2 = \emptyset$.

Definition 3.5. (Utility of a team for task execution) Let $\Gamma = \{x_1, \dots, x_k\}$ be a team that can execute a task $\tau = \langle \nu, l, t, k, \Psi \rangle$ where each member of the team was located at loc_{x_i} . The utility of a team Γ for executing τ is $\mathcal{U}_{\langle \Gamma, \tau \rangle} = -cost_{\langle \Gamma, \tau \rangle}$, where $cost_{\langle \Gamma, \tau \rangle} = \sum_{x_i \in \Gamma} \mu_{\langle x_i, \tau \rangle}$ and $\mu_{\langle x_i, \tau \rangle} = \mathbf{p}(x_i, \tau) \times \frac{1}{\alpha_{x_i}} + \mathbf{d}(loc_{x_i}, l) \times \beta_{x_i}$.

where $\alpha_{x_i}, \beta_{x_i} \in (0, 1]$ denote remaining battery coefficient and battery consumption rate respectively of (a robot) x_i , $\mathbf{p}(x_i, \tau)$ is the price of x_i for τ , $\mathbf{d}(l_1, l_2)$ is the distance covered when moving from l_1 to l_2 .

A robot with higher α value ensures that it will not fail due to its more remaining battery backup. A robot with lower β value ensures that it will last for a longer period of time.

4. Distributed algorithm for cooperative transportation

Following assumptions are made for the study. Multiple robots are required for any task execution. A robot can execute at most one task at a time. Each robot has a unique identifier (id). A wireless network that is lossless, message delay is finite, data is not corrupted during transmission is considered. Messages are delivered in a FIFO manner.

Informal description of the algorithm is given below. Let a robot i attend a task $\tau = \langle \nu, l, t, k, \Psi \rangle$ where $\psi_i \supseteq \tau.\Psi$. To execute the task (cooperative transportation), initiator communicates with other robots in order to form a runtime team. Here, the i is named as an initiator, and the other robots as non-initiators.

After task detection, i broadcasts a *Request* message to know the current state of the other robots present in the environment at that moment in time and waits for some time, say Δ . The broadcast messages are delivered only to those robots who are present in the range. Now, on receipt of *Request* message, a non-initiator j takes the necessary actions. A non-initiator who has the desired skill will send a *Willing* and an *Engaged* message if its state is other than *Idle*.

The counter c of initiator is increased on receipt of *Willing* message. The parameter $(c \geq k - 1)$ is checked after Δ time has elapsed. Now, if the value of $(c \geq k - 1)$ is true then team formation that has maximum utility is possible and sends *Confirm* message to the members of the team and sends a *Not-Required* message to $(c - (k - 1))$ robots, if any. However, if the value of the condition $(c \geq k - 1)$ is false, i sends a *Not-Required* message to all c robots who expressed their willingness to help. Also, i changes its state from state *Ready* to *Idle*. The algorithm has the non-blocking property since a timer is used. If there was no timer, an initiator would have waited indefinitely and thereby forcing some non-initiators to wait indefinitely as well; thus the system would be blocked.

The *receive* function of a robot is given in Algorithm 2. The agents take the action based on the current state that may be *Idle* (line 17–21), *Promise* (line 22–39), *Busy* (line 12–16 and 41–45), and *Ready* (line 1–11). Within a state, the type of message is checked and appropriate actions are taken. For example, if an agent receives a *Request* message in *Idle*, the identifier of the sender is enqueued, and *flag* is set to true; if it has appropriate skills then it sends the *Willing* message to the sender (initiator) and *flag* is set to false.

The behavior of the agent is captured with communicating automata (CA) [7] as shown in **Figures 2** and **3**. Moreover, this communicating automata is helpful in understanding and designing the algorithm.

Transitions in CA are very general form $\chi : \gamma$, where χ can either be an input a (send message $!m$, receive message $?m$), or a state condition g , or (a, g) , and γ can

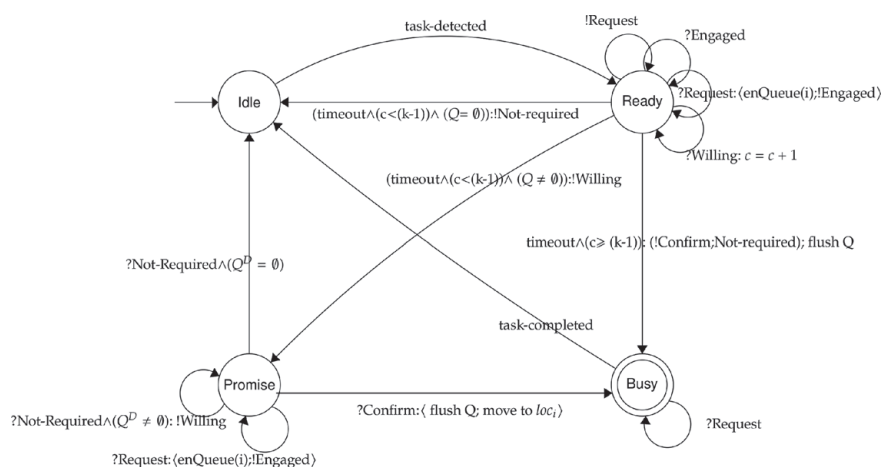


Figure 2.
 Finite state machine for an initiator agent.

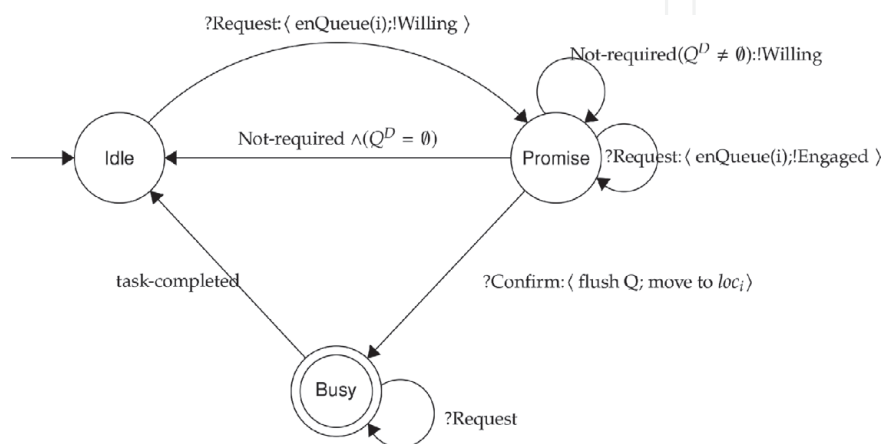


Figure 3.
 Finite state machine for a non-initiator agent.

either be a sequence of actions seq , or a sequence of actions that is to be performed atomically $\langle seq \rangle$, or empty. Similarly, semantics are defined.

4.1 Analysis of the algorithm

4.1.1 Message complexity

Let there be I initiators at some instant of time, say t . Each initiator broadcasts a *Request* message, which is sent to $(N - 1)$ robots, where N is the total number of robots present at time t . So, the total number of such messages would be $(N - 1) \cdot I$ which is $O(N \cdot I)$. The total number of replies obtained from non-initiators would be at most $(N - 1) \cdot I$ which is $O(N \cdot I)$. An initiator sends c number of *Confirm* and *Not-Required* messages, which is $O(N)$. Thus total messages send by all the initiators would be $O(N \cdot I)$. Thus the total number of messages would be the sum of these messages, and this becomes $O(N \cdot I) + O(N \cdot I) + O(N \cdot I)$, which is $O(N \cdot I)$. When the number of initiators is relatively small compared to the total number of robots present at time t , the message complexity would be $O(N)$.

4.1.2 Handling multiple initiators

Let us consider the snapshot of the environment at t_2 in **Figure 1**, where r_1, r_4 invoke the *send* function (Algorithm 1) simultaneously; r_1, r_4 need one, two other robots respectively. The initiators r_1, r_4 broadcast *Request* messages corresponding to their respective tasks. Let all the other robots be in *Idle* state initially and they can satisfy the requirements of both the tasks. Eventually r_2 becomes part of the team with r_1 because it received the *Request* message from r_1 before it received the *Request* message from r_4 . Similarly, eventually r_5 and r_6 become part of the team with r_4 .

Algorithm 1: (Send function of i)

```

Send function of initiator  $i$  for a task  $\tau = \langle v, l, t, k, \Psi \rangle$  :
1 state := Ready;
2 c := 0;
3 broadcast Request $i$ ;
4 set the timer and wait till timer is OFF;
5 if c  $\geq$  (k - 1) then
6   for each possible team, calculate utility as per Defn. 3.5;
7   Select the team that has highest utility, say  $\Gamma$ ;
8   send Confirm $i$  to the robots of the team  $\Gamma$ ;
9   send Not-Required $i$  to the other c - (k - 1) robots;
10   $\langle$ state := Busy; make Q $i$  empty $\rangle$ ;
11  initiate execution of task  $\tau$  when all members of  $\Gamma$  arrive at the location l;
12 else
13   send Not-Required $i$  to c robots;
14   if Q $i$  =  $\emptyset$  then
15     state := Idle;
16   else
17     state := Promise;
18     send Willing( $i$ ) to the front element of Q $i$ ;
19   end
20 end
21 if a Willing message is received in a state  $\neq$  Ready from a robot j then
22   send Not-Required $i$  to j;
23 end

Send function of Non-initiator j for a task  $\tau = \langle v, l, t, k, \Psi \rangle$  :
24 case Q $j$   $\neq$   $\emptyset$  and flag = true do
25   send Willing $j$  to the front element of Q $j$ ;
26   flag := false;
27 end
28 case Q $j$   $\neq$   $\emptyset$  and flag' = true do
29   send Engaged $j$  to the rear element of Q $j$ ;
30   flag' := false;
31 end

```

The complete execution trace of algorithms 1,2 is shown in **Figure 4** using message sequence chart (MSC).

Algorithm 2: (Receive function of j)

Receive function of Initiator j for a task $\tau = \langle v, l, t, k, \Psi \rangle$:

```

1  case state = Ready do
2  |   case msg = Request do
3  |   |    $\langle enqueue(Q_j, i); flag' := true \rangle$ 
4  |   end
5  |   case msg = Willing do
6  |   |    $c := c + 1$ ;
7  |   end
8  |   case msg = Engaged do
9  |   |   skip;
10 |   end
11 end
12 case state = Busy do
13 |   case msg = Request do
14 |   |   skip;
15 |   end
16 end

Receive function of Non-initiator  $j$  for a task  $\tau = \langle v, l, t, k, \Psi \rangle$ :
17 case state = Idle and  $\psi_j \supseteq \tau.\Psi$  do
18 |   case msg = Request do
19 |   |    $\langle state := Promise; enqueue(Q_j, i) \text{ if } i \text{ is not present in } Q_j; flag := true \rangle$ 
20 |   end
21 end
22 case state = Idle and  $\neg(\psi_j \supseteq \tau.\Psi)$  do
23 |   skip;
24 end
25 case state = Promise and  $\psi_j \supseteq \tau.\Psi$  do
26 |   case msg = Request do
27 |   |    $\langle enqueue(Q_j, i) \text{ if } i \text{ is not present in } Q_j; flag' := true \rangle$ 
28 |   end
29 |   case msg = Confirm do
30 |   |    $\langle state := Busy; \text{ make } Q_j \text{ empty; move to } loc_i \rangle$ 
31 |   end
32 |   case msg = Not-Required do
33 |   |    $dequeue(Q_j)$ ;
34 |   |   if  $Q_j = \emptyset$  then
35 |   |   |    $state := Idle$ ;
36 |   |   else
37 |   |   |    $flag := true$ ;
38 |   |   end
39 |   end
40 end
41 case state = Busy do
42 |   case msg = Request do
43 |   |   skip;
44 |   end
45 end

```

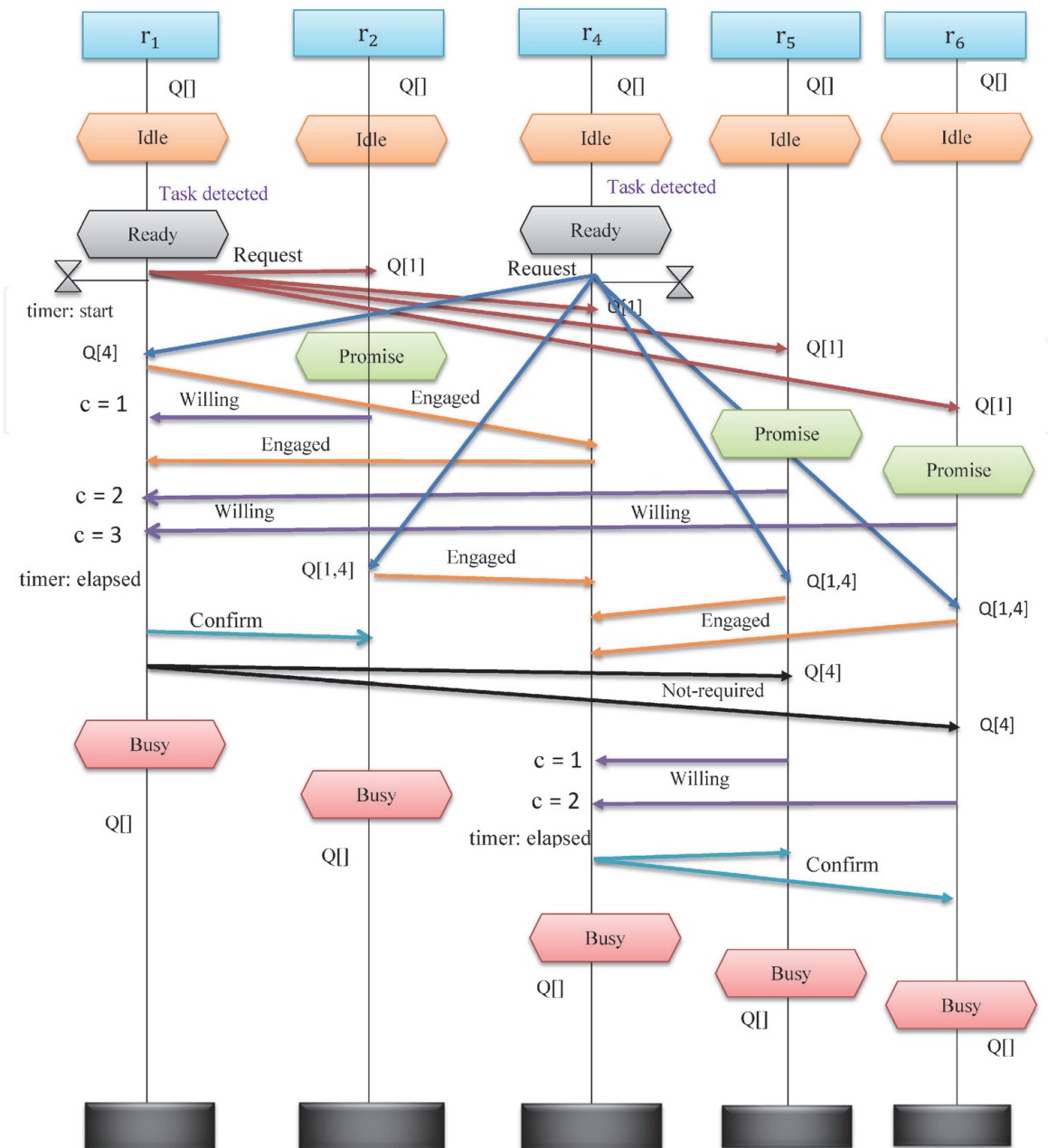


Figure 4.
Execution trace of the algorithms for multiple initiators.

5. Implementation in ARGoS

We consider a road clearance scenario to illustrate the proposed distributed algorithm (Section 4), where a road may be blocked by several obstacles. A team of robots should jointly move each obstacle to one side of the road. The algorithm is implemented using ARGoS (Autonomous Robots Go Swarming) [8], a multirobot simulator using the 3.0.0-beta47 version on Intel[®] Core™ i5 Processor, 4-GB of RAM and macOS Sierra operating system. The code run in ARGoS can be directly deployed on a real robot system.

An example scenario is shown in **Figure 5**, where the shaded portion in gray is the road (10 m × 5 m), obstacles are simulated by green movable cylinders of radius 0.2 m with a blue light on top. The robots are shown in blue. The overall process of removing an obstacle from the road is shown in **Figure 5**. The robots in ARGoS use the inbuilt range and bearing sensor (*rab*) to communicate among themselves.

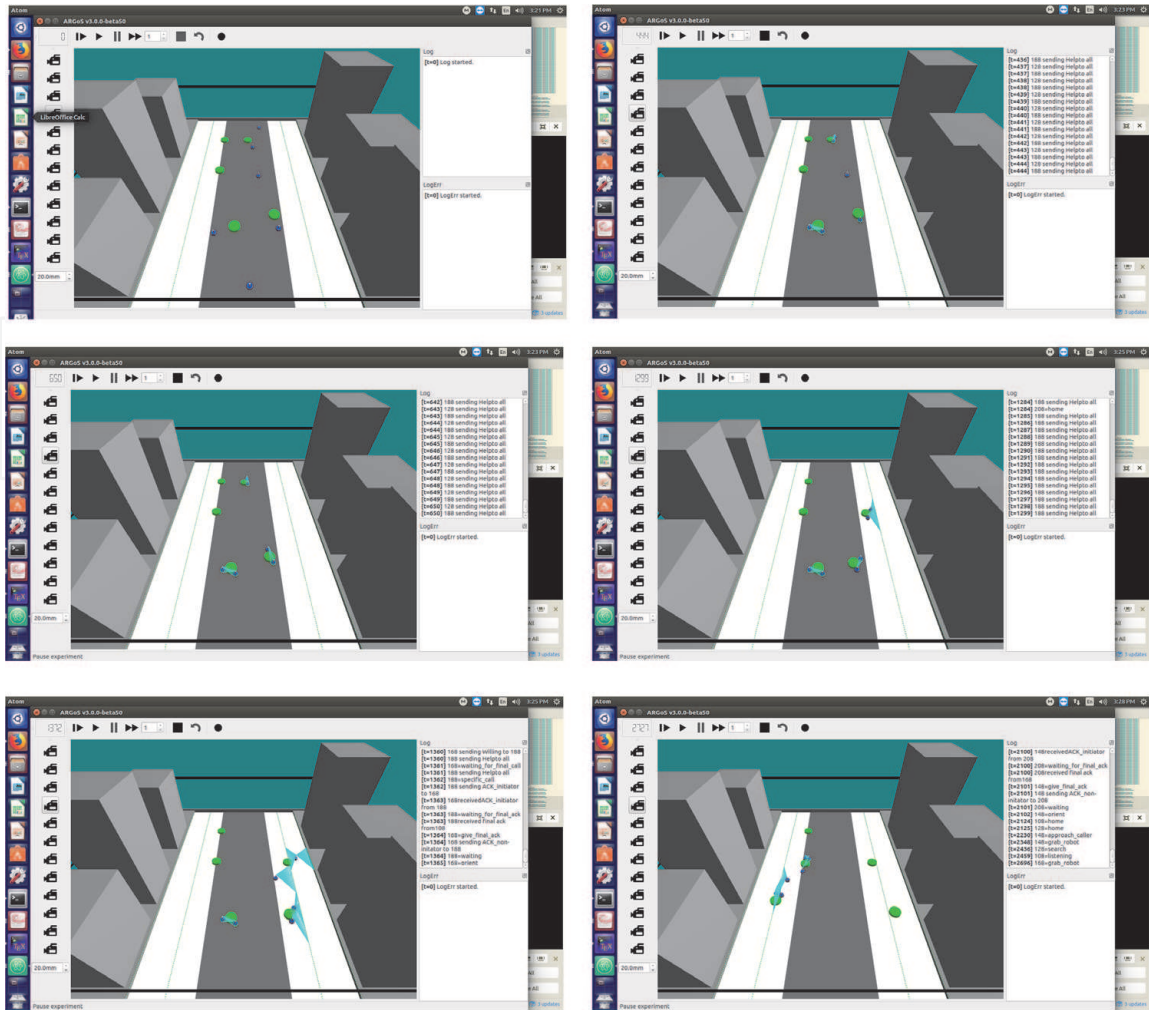


Figure 5.
 Illustration of multiple task execution in ARGOS.

The broadcast of the messages to all other robots is done by *rab* actuator. The broadcast of message is done within a certain range and in line of sight. We have used the 3 bytes for message within the range of 15 meters. The message is received by *rab* receiver within in the same network sent by *rab* sensors. Along with sending and receiving the message within range, *rab* sensors do the work of identifying the direction and distance from where the message is being sent. As the *rab* actuator allows the only broadcast, the address of the sender and that of the receiver needs to be specified in every message. Every robot in the simulation has a unique id of size 1 byte. Several sensors and actuators are used to control the movement and positioning of the robots. For example, proximity sensors are used to stay on the road and avoiding collisions with other robots, the omni-directional sensor is used to detect obstacles, gripper actuator is used to grip an obstacle, and turret actuator is used to turn the gripper actuator towards the direction of the obstacle.

In **Figure 5a**, the initial position of the robots and blocks is shown. Three robots detect the three obstacles and they start the formation for the same is shown in **Figure 5b**. We assume that all the obstacles require two robots to move. In **Figure 5c**, two initiator robots are able to form their teams. In **Figure 5d**, it is depicted that robots have reached to the location of obstacles and they are ready to move the obstacles. **Figure 5e**, clearly shows that both the obstacles have been shifted to one side of the road. After dropping the obstacles, the robots again visit the road and search for other obstacles if any. Finally, in **Figure 5f**, the third obstacle is also detected and removed. In this way, all the obstacles are removed from the road.

For the implementation we have written the required functions in Lua (a C-like language). These are: (i) to control the movement of a robot to avoid obstacle or another robot based on proximity sensor data, where the sensor detects an obstacle or another robot, (ii) control speed and velocity, (iii) synchronizing the robots for task execution, (iv) to control the movement of a robot when boundaries are detected using motor-ground sensors, (v) communication among robots based on the line of sight.

The implementation is carried out by writing the required function in Lua language. The different functions that are identified are as follows: (i) control of velocity and speed of the robot, (ii) control the movement of a robot so that obstacles and other robots could be avoided, (iii) synchronizing the robots in order to task execution, and (iv) communication among robots based on the line of sight.

6. Summary

Now, research in the field of robotics is going with a rapid rate. In many applications such as search and rescue, space, and automated warehouse, intelligent robots are being used. With the advancement of artificial intelligence domain, robots are becoming the good choice. A plenty of work has been carried out in the field of single robot. However, this chapter discuss the different aspects of work where multiple robots act on the same object at the same time. This problem becomes tough and different from normal multi-agent problem.

Cooperative transportation is common task in many challenging domains, i.e., rescue, mars and space, and autonomous warehouse etc. In this way the proposed framework becomes very much essential and important in such domains where multiple robots are required to execute a task.

The proposed approach also takes care of multiple task execution simultaneously, i.e., if multiple robots detect multiple different obstacles at the same time, the coalition formation process for each obstacle can be started. Each robot who detects the obstacle, starts the coalition formation, by executing the instance of the algorithms.

Author details

Amar Nath^{1,2*†} and Rajdeep Niyogi^{2†}


1 Sant Longowal Institute of Engineering and Technology, Punjab, India

2 Indian Institute of Engineering and Technology, Roorkee, India

*Address all correspondence to: amarnath@sliet.ac.in

† These authors contributed equally.

IntechOpen

© 2021 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. 

References

[1] Kube, C. R., & Zhang, H. (1993). Collective robotics: From social insects to robots. *Adaptive behavior*, 2(2), 189-218

[2] Mataric, M. J., Nilsson, M., & Simsarin, K. T. (1995, August). Cooperative multi-robot box-pushing. In *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, Vol. 3, pp. 556-561

[3] Gerkey, B. P., & Mataric, M. J. (2002). Sold!: Auction methods for multirobot coordination. *IEEE transactions on robotics and automation*, 18(5), 758-768

[4] Chen, J., Gauci, M., Li, W., Kolling, A., & Gro, R. (2015). Occlusion-based cooperative transport with a swarm of miniature mobile robots. *IEEE Transactions on Robotics*, 31(2), 307-321

[5] Kong, Y., Zhang, M., & Ye, D. (2016). An auction-based approach for group task allocation in an open network environment. *The Computer Journal*, 59(3), 403-422

[6] Gunn, T., & Anderson, J. (2015). Dynamic heterogeneous team formation for robotic urban search and rescue. *Journal of Computer and System Sciences*, 81(3), 553-567

[7] Brard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., & Schnoebelen, P. (2013). *Systems and software verification: model-checking techniques and tools*. Springer Science & Business Media

[8] Pinciroli, C., Trianni, V., OGrady, R., Pini, G., Brutschy, A., Brambilla, M., Dorigo, M. (2012). ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm intelligence*, 6(4), 271-295