The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

CoMutaR: A framework for multi-robot coordination and task allocation

Pedro M. Shiroma   Mario F. M. Campos

***Abstract*— In multi-robot systems, task allocation and coordination are two fundamental problems that share high synergy. Although multi-robot architectures typically separate them into distinct layers, relevant improvement may be expected from solutions that are able to concurrently handle them at the same "level". This paper proposes a novel framework, called CoMutaR (*Co*alition formation based on *Mu*lti-*ta*sking *R*obots), which is used for both tackle task distribution among teams of mobile robots, and to guarantee the coordination within the formed teams.**

Robot capabilities are modelled as actions, independent modules whose inputs do not depend on the robot that generated it. Solutions to tasks are devised as coalitions of actions, that can be spread amongst the available robots. We also define the concept of share-restricted resources, which are periodically checked and updated by the actions in the system. In contrast to prior approaches, this mechanism enables to quickly determine if two actions can be executed simultaneously, allowing a single robot to execute multiple tasks concurrently. A single-round auction protocol is used to automatically discover and form coalitions. Once a coalition is formed, coordination among robots is modelled as constraints imposed over the share-restricted resources. Finally, we have successfully implemented and applied CoMutaR in typical scenarios like object transportation, area surveillance, and multi-robot box pushing. Experimental results demonstrate that the system is able to provide good solutions even in the case of severe failures in participating robots.

I. INTRODUCTION

Multi-robot systems is a challenging field that has been receiving increasing attention in robotics over the past years ([1], [8], [9]). Among the advantages that such systems presents over single robots, we can highlight the increased robustness, efficiency, and higher range of tasks that can be accomplished.

However, if one wants such desirable features to appear in multi-robot systems, it is fundamental that such a system should be able to efficiently handle unforeseen issues. There are, at least, two main problems in multi-robot scenarios: (i) how to distribute tasks among robots, and (ii) how to coordinate their actions in order to successfully accomplish the assigned tasks. Although there is a significant synergy between these two problems, most of related work separate them into layers, in order to make them more tractable. However, it is expected that approaches able to handle both problems together may provide better solutions.

Also, if one expects robots to become ubiquitous in our society, such solutions must be able to handle the dynamical insertion and removal of possibly heterogeneous robots. According to the taxonomy proposed in [1], our work can be classified as a MT-MR-IA approach.

ALLIANCE [2] uses a distributed behavior-based architecture where tasks are performed by selecting a behavior set. Impatience and acquiescence attributes are used to trigger the process of taking over tasks from other robots or giving up one's own current task. M+ [3] is a decentralized protocol divided into three layers: A task allocator, based on the Contract Net Protocol [4]; a fault-tolerance module; and a task executor module, responsible for the coordination. However, all robots must receive the same mission description and there is low synchronization between the allocator and task executor modules. The CNP was also the starting point of several successful works, like MURDOCK [5], which uses a greedy algorithm and a time-limited contract to provide fault-tolerance, and Traderbots [6], a distributed architecture which form local centralized coalitions.

In multi-agent systems (MAS), a common concept present amongst most works is *coalition*, a temporary organization of agents that are brought together in order to solve a specific task. Theoretical results were conducted by [7] to study coalition formation on software agents, and were the inspiration of several works, like ASyMTRe [8], and RACHNA [9]. In ASyMTRe, robot capabilities are modelled as schemas, and all inputs and outputs are based in the information types [10] allowing them to share a common vocabulary. A task is defined as a set of motor schemas, for which solutions are automatically generated by forming a coalition of schemas. According to [9], a great number of works developed in MAS cannot be directly transferred to multi-robot scenarios, because they do not consider restrictions that arise with real robots, like lossy communication, device failure, situated agents, dynamical environments, and non-transferability of resources. Also, the position constraint imposed by physical devices implies that information produced has only local meaning, and an agent cannot immediately establish an anytime communication channel with other agents without having to reposition itself.

In [11] the task allocation problem is modelled as a hybrid automaton. Task assignment is treated as discrete events and the controllers are represented as continuous states. Therefore, it can potentially share the benefits of formal analytical machinery developed for hybrid automata.

A. Our approach

The development of our framework was based on the observation that, although any modern computer systems provide multitasking capabilities, only few robotic architectures are truly multitasking. One possible reason is that a robot cannot freely "change context" from one task to another without incurring in the possibility of wasting all the progress it may have reached up to that point in time. Hence, any attempt to allocate multiple tasks to mobile robots will face the spatial constraint, which is specific to the robotic scenario. Also, it is clear that no solution exists and multitasking is not feasible if two tasks require the same robot to be at different places at one given instant in time. However, this is not the general case, and there is a large number of tasks that impose less restrictive requirements to robot location, allowing for the sharing of several of the existing robotic resources present in a team.

For this class of tasks it is feasible to come up with solutions where a robot can execute two or more tasks simultaneously, if the *admissible configuration space* of the tasks overlap. This idea can be extended to a more general concept of share-restricted resources which embed, along with robot's pose, other features like processing power and communication bandwidth.

Our approach is based on the CNP to form coalitions of actions and is similar to the ASyMTRe [12] approach in many aspects. However, unlike [12], where the output of motor schemas are summed to generate the overall behavior, our approach tries to find out a motor output that will always be admissible for all current tasks. Our concept of action is strongly based on information invariants [10], sensor-databases [13], and schemas [14] which will allow us to divide a task into smaller robot-independent modules that can efficiently reuse past information gathered by the system and will require the production of new information.

However, our approach is distinct of others in that it enables the design of multiple concurrent coalition solutions, each one assigned to a task, which can include actions belonging to overlapping robots. This means that a robot can be assisting the execution of multiple tasks concurrently through the actions, and thus be classified as a multi-task (MT) robot. Also, our approach is able to handle the coordination problem that arises in such systems.

II. PROBLEM DEFINITION

Given $\mathbf{R} = \{r_1, r_2, ..., r_m\}$, a set of m heterogeneous mobile robots, $\mathbf{T} = \{t^1, t^2, ..., t^p\}$, a set of p tasks to be executed, which can be randomly inserted. Let $\tau_i \subset \mathbf{R}$ be a team of robots, and $\mathfrak{R} = 2^{\mathbf{R}}$ be the set of all subteams that can be formed. The problems addressed in this work can be stated as:

Problem definition 1 (Task allocation): Find a function $\mathcal{A} : \mathbf{T} \mapsto \mathfrak{R}$ such that $\mathcal{A}(t^k)$ is a team of robots capable of performing the task t^k.

Problem definition 2 (Team coordination): Given a set of robots $\mathbf{R}$, a set of tasks $\mathbf{T}$, and a task allocation $\mathcal{A} : \mathbf{T} \mapsto \mathfrak{R}$, coordinate the actions in τ_i during its persistence, such that they are able to accomplish the task t^i, and they do not interfere in the execution of the other tasks.

III. METHODOLOGY

A. Action

We model the robots capabilities or skills, such as `read laser`, `detect obstacles`, `avoid obstacles`, `push box`, as a set of *actions* which are, to some extent, similar to the *schema* concept [14]. Formally:

Definition 1: An action is any computational module that can either produce data, consume data, or accomplish a task.

We define $a_{i,j}$ as the j-th action in robot r_i. Also, n_i is defined as the total number of running actions in robot r_i. In order to avoid plurality of data types, we adopt, similar to [12], a set of information types $\mathbf{F} = \{f_1, f_2, ..., f_p\}$ and restrict all actions inputs and outputs as a subset of $\mathbf{F}$.

A key feature of defining the robot capabilities as a set of independent actions or schemas (like in [12], [15]) is that it makes it possible to transparently handle failures. Once there is no central planner, when a sensor fails, only the capabilities (actions) that depend on that sensor output are affected. Moreover, if another robot is able to provide the same information of the failed sensor, then higher level actions will still be able to be executed after the proper reconfiguration.

In contrast to approaches based on schemas, which only checks for the inputs to determine if they can be activated, we define sets of resources that has limited sharing possibilities. Resources like communication link, processor, battery power and the robot pose have physical restrictions that limit the amount of actions that can be concurrently running. For example, a communication link cannot exceed the device's maximum bandwidth, and hence, higher demands would not be acceptable. Therefore, our model should refuse new connections while the link is not capable to adequately handle new requests. We capture this concept by the following definition:

Definition 2: A share-restricted resource is any property in the environment that cannot be freely shared among the actions.

Examples of share-restricted resources are robot position, robot energy, communication bandwidth, and free configuration space. A share-restricted resource can either belong to a robot (e.g. energy, position) or be intrinsic to the environment (e.g. the free configuration space). Formally:

Let ${}^{i}\chi = \{{}^{i,1}\chi, {}^{i,2}\chi, \ldots, {}^{i,s_i}\chi\}$ be the set of share-restricted resources in the environment ($i = 0$) or in robot r_i ($i > 0$). For each share-restricted resource

${}^{i,k}\chi$, we associate a codomain ${}^{i,k}C$ which "measures" the availability of a share-restricted resource ${}^{i,k}\chi$.

We define, for each action $a_{i,j}$, a constraint function ${}^{l,k}\varphi_{i,j}(t) : \Re \mapsto {}^{l,k}C$ as a function of time that measures the amount of the share-restricted resource ${}^{l,k}\chi$ in robot r_l (or in the environment, if $l = 0$) required by $a_{i,j}$.

The space ${}^{l,k}C$ is defined such that it accepts two operators, a compound operator:

$$\oplus : {}^{l,k}C \times {}^{l,k}C \mapsto {}^{l,k}C, \tag{1}$$

which is used to "sum" the constraints imposed by two constraint functions, and a comparison operator:

$$\prec : {}^{l,k}C \times {}^{l,k}C \mapsto \{true, false\}, \tag{2}$$

used to check if the sum of constraint functions exceed the maximum capacity, ${}^{l,k}\varphi_{max}$, of the share-restricted resource. Define

$$\sum_{j=1}^{n_i} {}^{l,k}\varphi_{i,j} \triangleq {}^{l,k}\varphi_{i,1} \oplus {}^{l,k}\varphi_{i,2} \oplus ... \oplus {}^{l,k}\varphi_{i,n_i}, \tag{3}$$

as the constraint imposed by all running actions in robot r_i over share-restricted resource ${}^{l,k}\chi$ in robot r_l.

Similarly, define:

$$\sum_{i=1}^{m} {}^{l,k}\varphi_{i,j} \triangleq {}^{l,k}\varphi_{1,j} \oplus {}^{l,k}\varphi_{2,j} \oplus ... \oplus {}^{l,k}\varphi_{m,j}, \tag{4}$$

as the composition of the constraint functions imposed by all robots over share-restricted resource ${}^{l,k}\chi$. Thus,

$$\sum_{i=1}^{m}\sum_{j=1}^{n_i} {}^{l,k}\varphi_{i,j} \prec {}^{l,k}\varphi_{max} \tag{5}$$

can be interpreted as "Does the sum of the constraints imposed by all active actions exceed the maximum capacity of share-restricted resource ${}^{l,k}\chi$?"

Next we define the codomain, φ_{max}, and operators $\oplus$ and $\prec$ for the most common share-restricted resources in robotics.

1) Communication link: Communication, and specially wireless communication, has been fundamental for the operation of mobile robots. In real environments, packet loss, data corruption and network disconnection are usual events. Also, bandwidth limitations of wireless interfaces may make it difficult to transmit large volumes of data such as a streaming video.

Bandwidth is the main limitation imposed by communication links, so we define ${}^{l,k}\varphi_{max}$ as the maximum bandwidth. Thus, the codomain for the communication bandwidth is the set of real numbers ($C \triangleq \Re$), and the constraint function $\varphi_{i,j}$ is the required bandwidth to action $a_{i,j}$ properly execute its operations. In this case, the $\oplus : \Re \times \Re \mapsto \Re$ operator is defined simply as the algebraic sum, and $\prec : \Re \times \Re \mapsto \{true, false\}$ operator is the "less than or equal to" operator. For example, suppose that action $a_{i,1}$ requires data size of 100 Kbits at a rate of 100 samples/sec (totalizing 10Mb/s), action $a_{i,2}$ requires 40 Mb/s, and action $a_{i,3}$ demands 30Mb/s. Thus, if the maximum bandwidth, φ_{max}, is $100Mb/s$ then actions $a_{i,1}$, $a_{i,2}$ and $a_{i,3}$ can share the communication resource since $10Mb/s + 40Mb/s + 30Mb/s \leq 100Mb/s$.

2) Processor: The dynamical aspect of the real world coupled with the complexity of data analysis of some data sources, like cameras, turn the robot into a voracious processor consumer. However, in order to increase energetic autonomy, a robot should rely only in low power components, which consequently will restrict the processing power of the system. The efficient use of the processor is essential for any architecture that will handle task allocation. Thus, if the processor limitations are not obeyed, it can cause degradation on the running process, specially in those that impose real-time constraints like low-level controllers.

We define φ_{max} as the maximum processing power allocated to our application. Since computation time for other processes must also be considered (e.g. operating system and user interfaces), φ_{max} will be, in general, smaller than the total processing power available. The constraint function $\varphi_{i,j}$ for an action $a_{i,j}$ is the required processing power for that action.

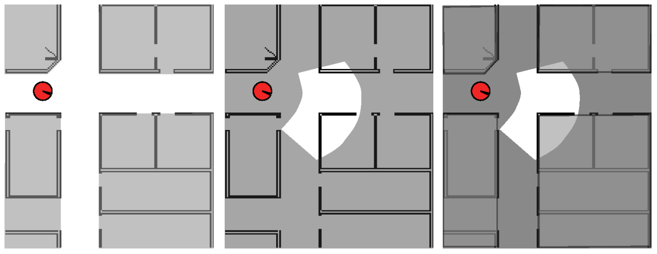
The operator $\oplus : \Re \times \Re \mapsto \Re$ for the processor may be defined as the algebraic sum, and the operator $\prec : \Re \times \Re \mapsto \{true, false\}$ is defined as the "less than or equal to" operator.

3) Position: One of the most important share-restricted resources that we must deal with in robotics is the robot position. The position of a robot is intimately related to the motor and actuator resource and for a given action, the commands sent to the actuators may cause one of three effects:

- Make the robot advance toward its goal;
- Be irrelevant to the execution of that action;
- Be harmful and provide a negative impact to the completion of the action.

For example, a surveillance action can define as virtual obstacles, in the configuration space, the whole area except the regions that it has to observe. In contrast, other actions can define more naturally its constraints in the velocity space [16], such as a controller based on potential fields. It is easy to see that, for a given configuration in $\mathcal{SE}(3)$, the mapping from velocity space to the Euclidean space can be computed with little effort. Also, note that, given a location in space, a potential field divides the space into two regions: one with smaller potential and another with larger potential. Other controllers can be easily adjusted by including a disturbance around the optimal output.

The codomain for the position resource is defined as the special Euclidean space ($C \triangleq \mathcal{SE}(3)$). The constraints of all actions can then be joined into the configuration space (Fig. 1) and, if their intersections is not the empty set, it means that there exist a set of motor commands that satisfies both tasks. Therefore, we have that $\oplus \triangleq \cap$, $\prec \triangleq \neq$ and $\varphi_{max} \triangleq \emptyset$. Remember that the



(a) ${}^{1,1}\varphi_{1,1}$ (b) ${}^{1,1}\varphi_{1,2}$ (c) ${}^{1,1}\varphi_{1,1} \oplus {}^{1,1}\varphi_{1,2}$

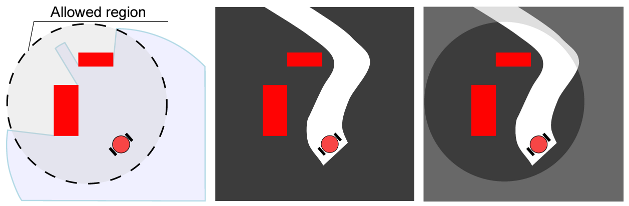
Fig. 1. In light gray the constraints imposed by a surveillance action. In dark gray the constraints of a controller. The intersection of the areas (in white) indicates the admissible velocities for both actions.

constraint function is a timed-function, so the region can change over the parameter t.

For example, in order to check if actions $a_{i,1}$, $a_{i,2}$ and $a_{i,3}$ can be executed concurrently, we test if $\varphi_{i,1} \cap \varphi_{i,2} \cap \varphi_{i,3} \neq \emptyset$ is true.

Note that not only actions that produce velocity commands can impose constraints over the position of a robot, but actions that read sensors can also restrict the allowed configuration space.

Suppose, now, that an action is querying the obstacles in a given region (Fig. 2). In order to properly answer the query, and consequently keep the established contract, the robot must stay at a maximum distance from the region (assuming an omnidirectional sensor).



(a) Constraint imposed by an omnidirectional sensor. (b) Constraint imposed by a path following action. (c) Composition of both constraints.

Fig. 2. A sensor can also constrain the allowed position of a robot. When the robot reaches the upper boundary of the area covered by the sensor, it must decide which action it will continue to execute and which one it will stall.

Table I summarize the constraint functions for the analyzed share-restricted resources.

Therefore, constraint functions allow us to evaluate if a set of actions can be concurrently executed by a robot.

B. Query

With the advent of multi-robot systems, many concepts that researchers have inherited from the early years need to be revisited in order to adapt to a new reality. One such a concept is the way information flows from sensors to actuators. While each robot can rely only in its own sensors, it would be more efficient if it could share the gathered information to other robots which sensors are presenting noisy data, malfunctioning, or being occluded. In this section, we extend the idea of sensor database [13], which treats sensors as databases that store all past information and whose data can be retrieved through the traditional use of queries. Unlike [13] we define for each database a metadata (m_i) that allows any module to check if the database has the necessary data for its execution. The metadata included in each database must be descriptive enough to allow the decoupling of the data from the source that produced. Consequently, producer and consumer are not required to be in the same physical robot and thus we can split the actions through the robots.

We restrict our actions such that the following metadata are enough to determine if an action can be executed or not:

- *type:* The class of information type ($\mathtt{type} \in \mathbf{F}$) being accessed.
- *id:* A list identifying the targets, it can assume a single value, a multiple list, or refer to any object that matches the other fields.
- *position:* A spatial area indicating where the data was produced.
- *time:* A time interval indicating when the data was produced. Reactive tasks will require short intervals which may be gradually extended if the environment is more static in nature. Deliberative tasks could rely on longer intervals.
- *rate:* This field indicates the required output data rate, making it useful for reactive tasks.
- *duration:* The duration of the query can be instantaneous, which means it will be answered only once, or time-extended.
- *error:* The maximum admissible error in the sent data.

A query in the resource database has the following syntax:

$$\sigma_{type,id,time,position,rate,error,duration}$$

Examples of queries are:

- $\sigma_{OBSTACLES,ANY,[0:-1s],[x0:x1,y0:y1],15Hz,1cm,20s}$
  This query could be requested by a robot that is trying to avoid obstacles while moving toward a waypoint. After the query is accepted, the resource database will continue to output data for 20 seconds. Any module that accepts to answer the query should plan its action to ensure that all requirements will be met or otherwise not accept it.
- $\sigma_{POSE,obj1,[0:-1s],ANY,15Hz,1cm,1min}$
  An action trying to track and pursue and object could use the query above.

Since we restrict the output of an action to be a predefined information type and the date rate is defined by the contract established by the query, we can determine the ${}^{comm}\varphi_{i,j}$ for all actions based on the rate and data type fields. Also, for actions that produce data, the position

TABLE I
Definition of constraint functions for most common share-restricted resources.

		Communication	Processor	Position
Codomain		$\Re$	$\Re$	$\mathcal{SE}(3)$
constraint function	φ	required bandwidth	required proc. power	feasible region
maximum resource capacity	φ_{max}	maximum bandwidth	allocated processing time	empty set
compound operator	$\oplus$	algebraic sum ($+$)	algebraic sum ($+$)	intersection operator $\cap$
comparison operator	$\prec$	algebraic comparison ($\leq$)	algebraic comparison ($\leq$)	difference operator $\neq$

and duration field in a query are sufficient to specify the ${}^{pos}\varphi_{i,j}$.

Actions and databases form a pair with all data flowing from an action to a database, located in the same robot, and then be exported to other actions. This allow us to query an action, not only on its current output, but also on previous data produced.

C. *CoMutaR - A multi-robot task allocation and coordination approach*

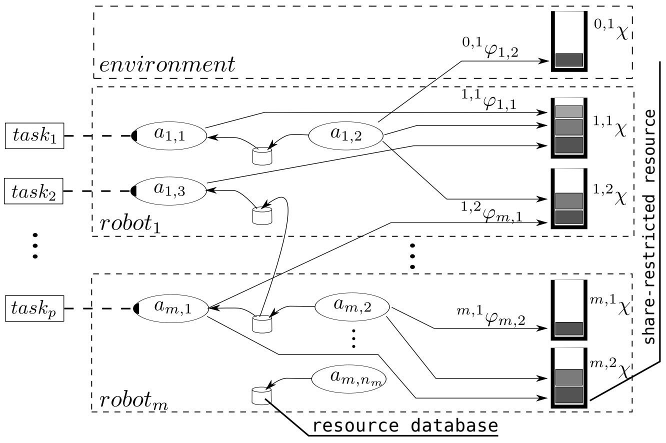


Fig. 3. Architecture overall. Each robot r_i has a set of actions ($a_{i,j}$) and a set of share-restricted resources (${}^{i,k}\chi$). The active actions can impose a set of constraint functions (${}^{l,k}\varphi_{i,j}$) over all resources. Coordination is represented as the constraint functions imposed to resources in other robots.

The motivation to work on this problem came from [1] where it is said that "the MT-SR-IA and MT-SR-TA problems are currently uncommon, as they assume robots that can each concurrently execute multiple tasks."

In this section we describe the *CoMutaR* (*Co*alition formation based on *Mu*lti*ta*sking *R*obots) approach (Figure 3 shows the main components and its relationship). Since it is known MRTA to be $\mathcal{NP}$-hard [1], the developed solution will not focus in finding an optimal solution. We define a coalition as:

Definition 3: A coalition is a temporary organization of actions that are brought together to tackle a particular task.

Then, a coalition of actions is assigned to each task. One key difference between the coalition concept adopted in this work and other approaches, is that here coalitions are formed by actions and not by schemas. Thus, we are not restricted to behaviors and we do not "add" the contributions of each motor schema to produce an output that, hopefully, will make all behaviors progress. The share-restricted resource concept guarantees that the output produced will always have a positive effect for all actions.

It is expected that an agent will insert tasks into the environment with a minimum interval of Δt. When a task is injected, the CoMutaR framework will handle it in two stages. In the first stage, an auction is opened and the set of all coalitions that can handle the task is generated. Our approach is based on the CNP [4] and uses a one-round auction process. Among the advantages of such systems, we can highlight its scalability, fault tolerance and efficiency. We assume all robots to be truthful and their bids reflect the real (or expected) revenue. Each action capable of performing the task starts the data connection process, followed by a bidding. If the inputs of an action are not connected, a query is broadcast to find a suitable connection. This process continues until all inputs are connected (which means that a coalition can be formed), or if an input cannot be connected (meaning that this coalition cannot run). In order to synchronize the negotiation flow, each subsequent query has a shorter time interval to be answered than its predecessor. Next, all potential coalitions send a bid, taking into account the costs of all elements in the coalition.

In the second stage, the auctioneer determines a winner and announces it. The auctioneer can vary from task to task and can be either a client connected to a graphical interface or another robot. The winner coalition adjusts its share-restricted resources and the proper data connections are established. Figure 4 shows the execution flow when a task is announced in the system. In this example, action $a_{1,1}$ requires a data that can be produced by either $a_{1,2}$ or a_{m,n_m}.

After the winner is announced, the coalition starts to execute the task and a new task can be inserted in the system. Internally, each action runs two algorithms: one to respond to the request of new tasks and to form coalitions (Alg. 1), and another algorithm to respond queries, similar to Algorithm 1, but with tasks being replaced by queries.

D. *Robot coordination*

Robot coordination is modeled as a constraint function ${}^{l,k}\varphi_{i,j}$, where $i \neq l$, i.e., actions allocating resources

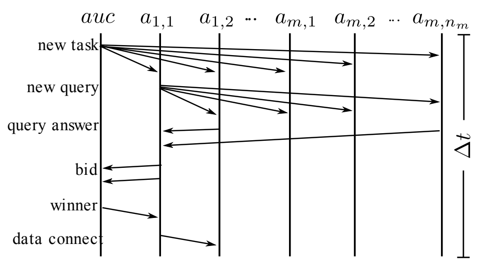


Fig. 4. When a task is announced, each capable action tries to connect its inputs through the use of queries. Similarly, each action capable of answering the query tries to connect its inputs, and this process continues until all inputs can be connected (which means a coalition can be formed), or one input can not be fed. In the example above, two coalitions ($\{a_{1,1}, a_{1,2}\}$ and $\{a_{1,1}, a_{m,n_m}\}$) are formed.

Algorithm 1: Bidding algorithm in robot r_i

```
Input: a task $t^k$
Output: a set of coalitions $\mathbf{C}$
$\mathbf{C} = \emptyset$
foreach $a_{i,j} \in r_i$ that can accomplish the task $t^k$ do
    for $l \leftarrow 1$ to $m$ do
        for $k \leftarrow 1$ to $s_l$ do
            if ${}^{l,k}\varphi_{i,j} \oplus \sum_{p=1}^{m} \sum_{q=1}^{n_p} {}^{l,k}\varphi_{p,q} \not\prec {}^{l,k}\varphi_{max}$ then
                return $\emptyset$
    if inputs of $a_{i,j}$ are disconnected then
        send a new query
        wait for the query answer
        forall query answers do
            $c$ = coalitions in the query answer
            $\mathbf{C} = \mathbf{C} \cup c$
    else
        $\mathbf{C}$ = current coalition
return $\mathbf{C}$
```

in other robots. For example, suppose that we have an action $a_{1,j}$ that wants to send data from a robot r_1 to a base station through an ad-hoc wireless network, passing through robot r_2 (Fig. 5). To accomplish this task, $a_{1,j}$ has to constrain ${}^{2,pose}C$ to avoid it from breaking the communication link, and ${}^{2,comm}C$ must be allocated with the size of the transmitted data.

E. Assumptions

In summary, we have that if the following assumptions hold:

- If a robot is executing a task and it stops, the overall progress for that task does not decrease (all actions move toward the completeness of the task);
- A task can be subdivided into a set of actions, and each action has inputs and outputs clearly defined;
- Given a query, an action is able to determine if it can successfully provide all the required information or not;
- An action can be activated only if all inputs are available and its constraint functions do not exceed any share-restricted resource;

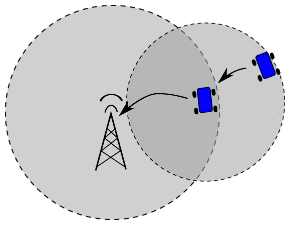


Fig. 5. Coordination among robots arises when one robot interfere in the share-restricted resources of another robot. In this example, a robot is serving as a gateway to the data collected by another robot, which is constraining both the available position and communication bandwidth.

- All robots are trustworthy and their bids reflect the real (or expected) revenue;
- There is a minimum interval (Δt) between the insertion of two consecutive tasks (IA assignment);
- The environment presents a perfect communication model;
- When a device fails, it stops the data production.

then the proposed framework is able to allocate the tasks and coordinate the robot actions.

IV. EXPERIMENTAL RESULTS

A. Simulation setup

To validate our architecture we designed two experiments: (i) a tightly-coupled task where two robots cooperate to push a large box, and (ii) a set of three tasks executed by two robots, namely, transportation, obstacle avoidance, and surveillance. The experiments were conducted on the player/stage/gazebo [17] simulator, but each robot is controlled by an independent software client, which is responsible for listening for new tasks, bid for new auctions, query for new data, and execute the assigned tasks. All communication among clients (i.e. robots) is implemented using the TCP protocol, and the only centralized portion of it is a custom broadcaster, responsible to assign ID's and relay multi-cast messages. The available actions to the robots are shown in Table II. Note that some actions require more than one input. The data-exporter and data-importer are capable of producing the same type of data present at the input, but with respect to a different frame. Although similar to schema-based approaches [12], [15], in our approach, the sum of motor outputs is replaced by the position share-restricted resource.

B. Box-pushing

The box-pushing task consists of moving a large rectangular (blue) box for 3 meters in a given direction, so that the box achieves a desired final pose. The large size of the box enforces that at least two robots must coordinate their actions in order to successfully accomplish the task. Initially, the two robots are facing the same side of the box. The position of a box relative to a robot is given by the `detect box` action, which consumes

Action	input		output
push box	box-pose	robot-pose	–
detect box	range-data		box-pose
read laser	–		range-data
localize	–		robot-pose
export data	*	robot-pose	*
import data	*	robot-pose	*
survey	robot-pose		–
transport	robot-pose		–
avoid obstacle	obstacles		–
detect obstacle	range-data		obstacles

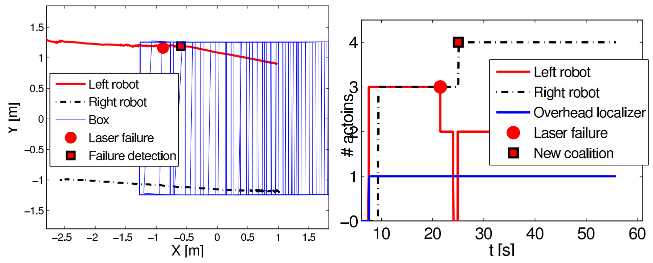
TABLE II

List of available actions.

data produced by a `read laser` action. The `push box` actions generate the control commands based on the box position read earlier, and the robot position produced by a `localize` action.

Fig. 6(a) shows the trajectories of the two robots and the box. In order to show how our system handles failures, we intentionally halt the laser sensor of the left robot. Fig. 6(b) shows the number of running actions in each robot. Note that when the sensor fails, a new configuration is formed in order to provide the missing information to the robot on the left. This robot starts to consume the box position produced by the right robot (a new `export data` action starts to run, and a `import data` replaces the halted `detect box`).

Note that the trajectory is smoother, and both robots proceed on executing the task, which is an improvement over others systems.



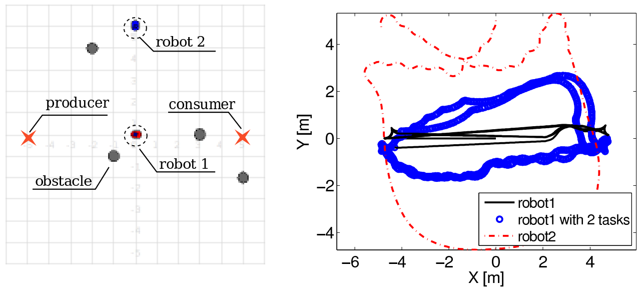
(a) Box and robots trajectories. (b) Number of running actions in each robot.

Fig. 6. A box pushing task being executed by two robots. Each robot uses its laser range-finder to localize the box and compute the motor outputs. A sensor failure is injected in the left robot in the middle of the execution.

C. Transportation and Surveillance

In the context of this experiment, a transport task is defined as *waypoints* (producer and consumer) that a robot must continually and sequentially visit. For instance, the producer waypoints could be defined as a set of objects in the environment that have to be disposed and the consumer waypoint as the waste deposit. In this experiment we set the producer to be at $(-5, 0)$ and the consumer at $(5, 0)$ (Fig. 7(a)). Figure 7(b) shows the trajectory described by the robot during task execution.

To illustrate our system's capacity of handling multiple tasks simultaneously, a second task is inserted after



(a) Experimental setup consisting of two robots in an environment with obstacles. (b) Trajectories described by the robots.

Fig. 7. Robot 1 executing multiple tasks simultaneously.

awhile. The second task consists of surveying an 6m × 6m area around the origin. In order to accomplish this task, the robot must pass, at least once, closer to all points within the region. The task is allocated to the first robot, which adjusts its share-restricted resources (Fig. 8). The amount of available resource is estimated considering only the positions that can be reached by the robot up to the next iteration. The surveying task is defined such that it allows the robot to be at any position in the environment, and thus its constraint function does not affect in the available positions. However, it does affect the determination of the velocity that will be used. Therefore, in order to accomplish all tasks, the robot starts to move off the line connecting the waypoints defined by the transportation task. It proceeds until it reaches an equilibrium and the robot is no longer capable of executing the surveillance task adequately, and eventually it goes on to watch previously visited areas. At this point in time, the robot detects the lack of progress and re-opens the task so it is executed over the remaining area. Then, Robot 2 wins the second auction and proceeds to cover the unvisited area.

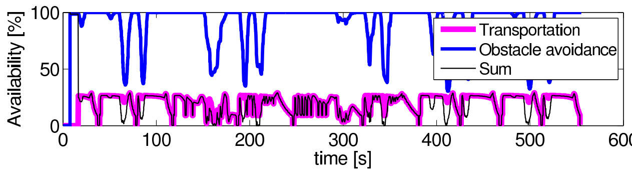
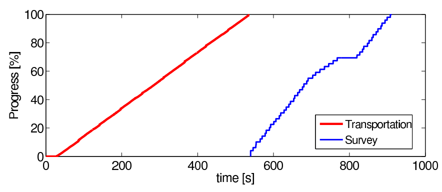
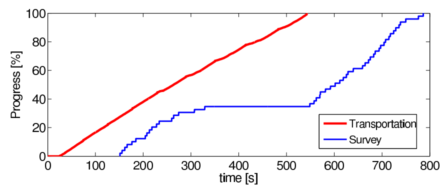


Fig. 8. Availability of the position share-restricted resource for Robot 1 while transporting, surveying, and avoiding obstacles. The survey action does not restrict the position share-restrict resource, but it does interfere with the choice of velocity.

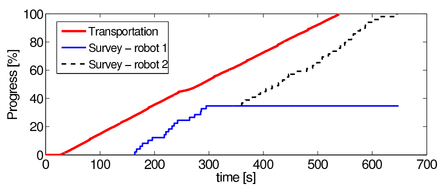
Figure 9(a) shows the progress of the two tasks being accomplished separately, which is how previous work would accomplished them. In a second scenario, we use the proposed framework and the robot execute both tasks simultaneously (Fig. 9(b)). Notice that although the completion time for the first task has increased 10 sec, the overall completion time has decrease from 908 to 786 sec). Finally, we allow the robot to give up the assigned task when it detects the lack of progress allowing



(a) A single robot executing two tasks sequentially.



(b) A single robot executing two tasks concurrently.



(c) Two robots executing two tasks.

Fig. 9. The progress of the tasks in three different scenarios.

a second robot to finish it (Fig. 9(c)).

V. CONCLUSION

We have presented a novel architecture, called CoMutaR, that is able to tackle, among several typical task allocation issues, two important problems in multi-robot systems: the multi-robot task allocation and coordination. The major contributions of CoMutaR are (i) the development of a system that enables robots to perform multiple tasks concurrently, and (ii) to bring both the requirements necessary to allocate tasks, and the constraints imposed by the tasks during its execution, into the same consistent framework.

Experimental results were performed using the player/stage/gazebo simulator in both loosely-coupled tasks like area surveillance and transportation, and tightly-coupled tasks like box pushing, and the results have shown that our framework was able to successfully resolve the required allocation issues.

The introduction of share-restricted resources and constraint function concepts enabled us to clearly define when two tasks can be concurrently executed or when a robot can form a new coalition. It also allows us to design a system that is robust to failures and which presents better performance when compared to similar approaches in the literature. The absence of a central planner makes CoMutaR particularly well suited for multi-robot environments.

As part of the ongoing work, we plan to study the impact of imperfect communication on the architecture. This can be modelled as a more complex communication share-restricted resource, which may involve, for instance, devising different policies that will assist in selecting the best velocity. We are also experimenting with different types of environments and with a significantly larger number of robots.

A. Acknowledgments

The authors would like to thank CNPq, CAPES, and FAPEMIG for their support.

References

[1] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The Intl. Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, September 2004.

[2] L. E. Parker, "Alliance: An architecture for fault tolerant multi-robot cooperation," *IEEE Trans. on Robotics and Automation*, vol. 14, no. 2, pp. 220–240, April 1998.

[3] S. S. C. Botelho and R. Alami, "M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, vol. 2, Detroit - Michigan, May 1999, pp. 1234–1239.

[4] R. G. Smith, "The contract net protocol: high-level communication and control in a distributed problem solver," *IEEE Transactions on Computers*, vol. C-29, no. 12, pp. 1104–1113, December 1980.

[5] B. P. Gerkey and M. J. Matarić, "Sold!: Auction methods for multi-robot coordination," *IEEE Trans. on Robotics and Automation*, vol. 18, no. 5, pp. 758–768, October 2002.

[6] M. B. Dias, "TraderBots: A new paradigm for robust and efficient multirobot coordination in dynamic environments," Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January 2004.

[7] O. Shehory and S. Kraus, "Methods for task allocation via agent coalition formation," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 165–200, April 1998.

[8] F. Tang and L. E. Parker, "ASyMTRe: Automated synthesis of multi-robot task solutions through software reconfiguration," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Barcelona, Spain, April 2005, pp. 1501 – 1508.

[9] L. Vig and J. A. Adams, "Multi-robot coalition formation," *IEEE Trans. on Robotics*, vol. 22, no. 4, pp. 637–649, August 2006.

[10] B. R. Donald, J. Jennings, and D. Rus, "Information invariants for distributed manipulation," *The Intl. Journal of Robotics Research*, vol. 16, no. 5, pp. 673–702, 1997.

[11] L. Chaimowicz, M. F. M. Campos, and V. Kumar, "Dynamic role assignment for cooperative robots," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Washington - DC, May 2002, pp. 292–298.

[12] L. E. Parker and F. Tang, "Building multirobot coalitions through automated task solution synthesis," *Proc. of the IEEE*, vol. 94, no. 7, pp. 1289–1305, July 2006.

[13] A. Cowley, H.-C. Hsu, and C. J. Taylor, "Distributed sensor database for multi-robot teams," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, vol. 1, New Orleans, LA, April 2004, pp. 691–696.

[14] R. C. Arkin, "Motor schema based navigation for a mobile robot: An approach to programming by behaviour," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, vol. 4, March 1987, pp. 264 – 271.

[15] L. Vig and J. A. Adams, "Coalition formation: From software agents to robots," *Journal of Intelligent and Robotic Systems*, vol. 50, pp. 85–118, 2007.

[16] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine*, vol. 4, no. 1, pp. 23–33, March 1997.

[17] T. H. Collett, B. A. MacDonald, and B. P. Gerkey, "Player 2.0: Toward a practical robot programming framework," in *Australasian Conference on Robotics and Automation*, Sydney, Australia, December 2005.