Decision making in multi-UAVs systems: Architecture and Algorithms

Simon Lacroix, Rachid Alami, Thomas Lemaire, Gautier Hattenberger and Jérémi Gancet

LAAS-CNRS, 7 avenue du colonel Roche, 31400 Toulouse
firstname.name@laas.fr

Summary. This chapter depicts an architecture that aims at designing a multi-UAV framework enabling cooperative operations in a system in which some UAVs are directly controlled by an operator, others are only endowed with *operational autonomy*, and others have *decisional autonomy* capacities. The architecture provides with the possibility to configure dynamically the decisional scheme, depending on the available robots and on the operational context.

A taxonomy of robots decisional autonomy is introduced, and used as a foundation to state the proposed architecture. The various functionalities on-board each robot are organized among a repartition that exhibits on-board functional components, and on-board or on-ground generic executive and decision making processes.

A set of algorithms that fulfill the three main decision-making functionalities required in a multi-robot system are then presented: a contract-net protocol that can handle task allocation for complex multi-UAV missions, a planning scheme based on a Hierarchical Task Networks planner completed with plan-refiners that consider the actual domain models, and an executive system that handles the coordination and task execution issues.

1 Introduction

Several UAV projects have been led by different research teams. Many are focused mainly on the development of advanced flight control abilities, *e.g.* to achieve aggressive maneuvers with helicopters, and others rely on operational UAV autonomy, *i.e.* the UAVs receive a pre-planned sequence of tasks to achieve, and do not exhibit high level planning or decisional skills – a noticeable exception being the Witas project at Linköping University [1]. When it comes to multi-UAVs, various topics have been studied. In the Anser project [2], the focus is set on the data fusion between multiple vehicles, *e.g.* to recover the positions of features on the ground [3].

Several contributions tackle the formation flight problem according to a control theoretic approach [4, 5], or using reactive behavior-based controllers in [6]. For this problem, the deliberative activities consist in determining the

trajectories to be followed by the formation (according to either a centralized [7] or distributed paradigm [8]), in selecting the internal formation geometric configuration [9], and in achieving switches between two given geometric configurations [11].

Fewer contributions deal with multi-UAVs problems according to a deliberative paradigm, where the UAVs exhibit cooperative and coordinated activities produced by high level planners, while maintaining reactive abilities during the execution of the plans (*e.g.* [10, 12, 13]).

Problem statement

We are interested here in the deployment of a multi-UAVs system to achieve observation missions. These missions can consist in detecting particular events over a given area, in monitoring their evolution, or in surveying (mapping) a given area. The system is *controlled by the operators*, in the sense that it achieve missions decided by operators according to their needs and the current knowledge of the situation they have access to. In particular, depending on the situation, a human operator should be able to handle the control of any UAV, at *any control level*. This implies that the operators should be able to specify high level missions, elementary tasks (such as setting a waypoint to reach by a particular UAV), or could even directly control the motions of an UAV. This is an essential feature required in any multi-UAV application context, where the operators need to master the activities of the whole system. Of course, this does not preclude the autonomous achievements of elementary tasks or high level missions: tedious and repetitive operations such as surveying an area relieve the operators if they are performed autonomously. There are even cases where only an advanced planning and control system is able to efficiently succeed, *e.g.* the coordinated operations of two or more UAVs flying out the operators line of sight is extremely difficult to control remotely.

The main consequence of this "controlability" requirement is that the system must be able to integrate UAVs with various levels of autonomy, from simple teleoperated UAVs to UAVs endowed with mission planning abilities. In other words, the system must enable both centralized (*i.e.* human-centered, via a central ground station) and distributed (*i.e.* delegated to UAVs) configurations of the decision. This significantly influences the design and implementation of the decisional architecture and algorithms introduced here.

Approach and outline

The overall system architecture follows a classic ground station / flying segment scheme. The central ground station is endowed with all the necessary monitoring facilities (in particular, it can include a data processing module that help the operators to assess the situations). It also has some UAV control means, and mission design and planning abilities.

The ability to integrate *heterogeneous UAVs*[1] and to exhibit adjustable autonomy is mainly brought by the design of the UAV on-board architecture and associated tools. Section 2 depicts this architecture, that provides with the possibility to configure dynamically the overall decisional scheme, depending on the available robots and on the operational context. A taxonomy of robots decisional autonomy is introduced, and used as a foundation to state the proposed architecture. It incrementally describes increasing schemes of autonomous decision-making, ranging from no decision-making capabilities up to autonomous task planning capabilities, autonomous coordination and even autonomous dynamic task re-allocation among the UAVs.

Many single UAV missions can be achieved only with operational autonomy abilities: the UAV receive a pre-planned sequence of tasks to achieve, and do not need high level decisional skills. But multi-UAV systems involve more temporal constraints and higher uncertainties on tasks execution: they can require higher autonomy abilities, ranging from coordinated execution control to task allocation. Sections 3 to 5 presents a set of algorithms that enable various levels of autonomous abilities :

- section 3 depicts an executive system that handles the coordination and task execution issues,
- section 4 presents a mission planning scheme based on a Hierarchical Task Networks planner completed with plan-refiners that consider the actual domain models,
- and section 5 presents a distributed allocation scheme based on a contract-net protocol, that can handle complex multi-UAV missions with temporal constraints between tasks.

2 UAV architecture

Numerous robots architectures have been proposed in the literature. Subsumption architectures [15, 16], exhibit behavioral robots models built upon reactive capabilities, whereas layered models explicitly divide up robots capabilities, as in the three-layers architectures (deliberative, executive and functional) [17, 18], or the 2-tiered CLARATy architecture [19]. Multi-robot architectures embrace additional concerns: designing a multi-robot architecture requires the definition of the decision making scheme and the specification of the interaction framework between the different robots, which of course influences the definition of the individual robots architecture. ALLIANCE [21] provides a behavior-oriented solution, enabling the design of totally distributed, fault tolerant multi-robot systems, whereas Simmons & al. [20] extend the three-layers architecture model within a multi-robot framework, where interactions between robots may occur along the different layers.

[1] the word “heterogeneous” being essentially related to decisional autonomy capabilities

Each of these multi-robot architectures enables the coordination and cooperation of several robots, but assumes a given homogeneous level of decisional autonomy for all the robots. They can enable the integration of physically heterogeneous robots, but can not cope with heterogeneous robots in terms of *decisional capabilities*.

2.1 A taxonomy of decisional autonomy capabilities

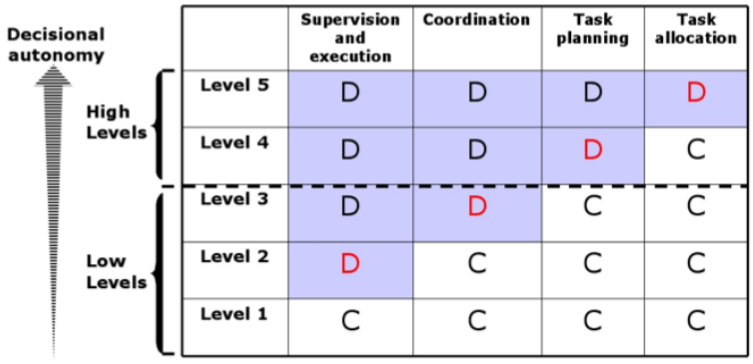
In any multi-robot system, decisional autonomy encompasses the following four abilities:

- Task allocation: How to distribute tasks among the robots ? This requires to define both a task assignment protocol in the system, and some metrics to assess the relevance of assigning given tasks to such or such robot.
- Mission refinement, planning and scheduling: How to transform a task or a mission into an executable sequence of actions ? These decisional activities are dedicated to plan building, considering models of the actions of the involved UAV, and of the environment.
- Coordination: How to ensure the consistency of the activities within a group of robots ? This requires the definition of mechanisms dedicated to prevent or solve possible resource conflicts (time and space resources), as well as mechanisms to plan and control the execution of joint cooperative tasks.
- Supervision and execution control: How to ensure that the planned tasks are correctly executed ? A system that manages the task execution and consider the encountered contingencies is required for that purpose.

These decisional components can be implemented according to different multi-robot systems configurations: they can be gathered within a central decisional node, or be partially (or even totally) distributed among the robots themselves. We define the "level of autonomy" of a robot as the amount of decisional components it is endowed with, and consider the following five levels (figure 1):

- **Level 1**: no autonomy on board the robot. The robot is only able to directly execute elementary tasks requested by the central decisional node.
- **Level 2**: executive capabilities. The robot is able to manage partially ordered sequences of elementary tasks, and to return execution status of the tasks.
- **Level 3**: same as level 2, plus coordination capabilities. The robot may manage on-line simple interactions (synchronizations) directly with other robots endowed with at least the same level of decisional autonomy.
- **Level 4**: distributed deliberative capabilities. High level tasks requests are managed (task planning and scheduling), and the multi-robot tasks coordination is autonomously ensured in a distributed way among robots endowed with at least the same level of autonomy.

- **Level 5**: same as level 4, plus tasks re-allocation capabilities. The robots may opportunistically re-allocate tasks and accept new tasks from other robots of the system endowed with the same level of autonomy.



	Supervision and execution	Coordination	Task planning	Task allocation
Level 5	D	D	D	D
Level 4	D	D	D	C
Level 3	D	D	C	C
Level 2	D	C	C	C
Level 1	C	C	C	C

Fig. 1. 5 levels of decisional autonomy. **C** stands for "Centralized", and **D** stands for "Distributed".

This taxonomy is characterized by a large gap between levels 3 and 4: up to level 3, a Centralized Decisional Node (CDN) is expected to ensure the global consistence of the system's activity: levels 1 to 3 are considered as **"low levels"** of decisional autonomy. Whereas levels 4 and 5 introduce the possibility to delegate coordination and mission refinement activities in a distributed way (**"high levels"** of decisional autonomy, embedded in the Distributed Decision Nodes – DDN), robots belonging to the fifth level being able to dynamically refine the allocation of tasks between them.

This taxonomy is to be understood in terms of *incremental delegation of decisional capabilities* by the multi-UAV system's user toward the UAVs. From the user's point of view, level 1 means a centralized full control of the system (*centralized* should be considered as available for operator). Level 2 enables an autonomous execution of partially ordered plan. Level 3 provides autonomous inter-UAV synchronization capabilities. Then a large gap appears between levels 3 and 4: up to the level 3, the CDN performs tasks planning and ensures the global consistency of the UAVs activities. Whereas level 4 is related to delegating mission refinement and planning activities to the UAV. Finally, level 5 enables autonomous tasks re-allocation : this is the highest delegation of decision making (i.e. the CDN only expresses high level goals to be achieved.

2.2 Decisional architecture

Figure 2 depict the overall architecture of the UAVs, for both low levels and high levels of decisional autonomy. A CDN communicates with robots, ex-

changing messages whose abstraction is defined according to the robots levels of autonomy. Each robot has a number of functional components, and is endowed with a generic Distributed Decisional Node (DDN) that enables various configurations of decisional autonomy, ranging from the simplest up to the highest decisional capabilities. It encompasses an executive (this executive being actually common to all levels, we denote it as the *Multi-Level Executive* - MLE), and a Deliberative Layer (DL) which provides robots with higher levels of decisional capabilities.

- **The Multi Level Executive.** For the low levels, the DDN is restricted to an executive. For level 1, the MLE behaves as a transparent connecting point between the CDN and the robot's functional components. For levels 2 and 3, it manages tasks sequences execution, and at level 3 it enables simple coordination interactions with other robots of the same level (these mechanisms are detailed in section 3). It acts in the same way for levels 4 and 5, the only difference being that it is interfaced with the UAV's DL instead of the CDN.
- **The Deliberative Layer.** For the high autonomy levels, the DL deals with missions and tasks refinements, coordination activities, and task reallocation (for level 5). It encompasses the following components (figure 2):
  - The symbolic planner builds flexible plans skeletons: it transforms high level missions requests into partially ordered plans. For that purpose, it uses the algorithms of the *specialized refiners* (4.3).
  - The specialized refiners gather a set of features to support tasks decompositions and refinements during planning and coordination, relying on the UAVs and the environment's models.
  - The interaction manager provides the means to coordinate UAVs activities, relying on distributed negotiation mechanisms, as Contract Net protocol to handle task allocations for instance.
  - The supervisor has a central position in the DL: it transmits missions refinement requests to the symbolic planner, then triggers negotiation sessions with the interaction manager in order to coordinate the resulting plans. It finally sends plans to be executed toward the MLE, and monitors returned tasks / plans execution status.

3 Execution control

We focus here on the functionalities of the Multi-Level Executive, using the context of an environment surveillance mission as a support example.

3.1 General task model and assumptions

The task model is built around elementary events processing: these events are expected to occur whenever the states of tasks evolves. Events can also

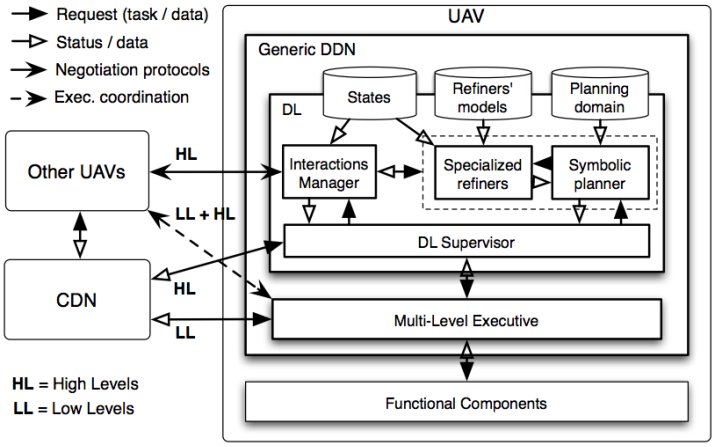


Fig. 2. DDN's components

correspond to other noticeable activities evolution, such as the reception of a message, or the elapsing of a certain amount of time. Tasks have a temporal extent: a task starts, then ends after a certain amount of time. The starting event is the only controllable event: all other kind of events related to a task are contingent, *i.e.* the system can not guarantee that such an event will occur, neither exactly when it may occur. A task can give rise to several, partially ordered contingent events during its execution.

For the low levels of autonomy, the Central Decisional Node (CDN) is supposed to be able to elaborate a safe and consistent multi-robot plan, and therefore to provide the robots with the tasks to be processed, according to a task communication formalism. On the other side, the minimal requirement expected from a robot is its capability to execute *elementary tasks*, *i.e.* unitary "simple" tasks that can be handled by robot's functional components. In the an environment surveillance mission, the following tasks are expected to be processable by a robot integrated in the system: take-off (TO), go-to (GT), take-shot (TS), wait (WT), and land (LD).

Integrating an UAV in the whole system requires to have this UAV endowed with a basic interface enabling elementary tasks information transmission (request, status, execution's result). For that purpose, an *elementary task formalism* has been developed (figure 3 – its specification is not necessary to detail here).

3.2 Executive's mechanisms

For the first level of decisional autonomy, the MLE is passive: it only transmits the elementary tasks requested by the CDN to the functional components of the robot, and sends back execution status.

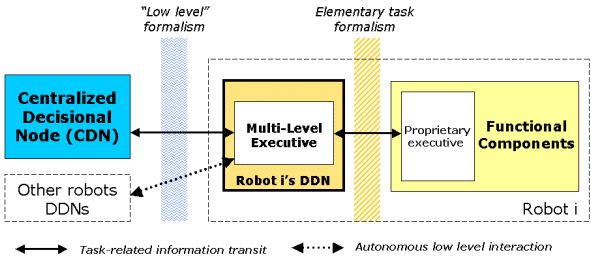


Fig. 3. Communication formalisms in the low decisional autonomy levels configurations

For the second decisional autonomy level, the MLE manages partially ordered sequences of tasks in a consistent way and in a timely and safe manner. Two main mechanisms are involved for this purpose:

- **dynamic tasks insertion:** this enables the possibility to request tasks insertion in the current task plan, according to an insertion mode that will characterize the relative order of the newly inserted task versus the current partial order of the already scheduled tasks. Four possible insertion modes are defined:
  - **SEQuential (SEQ) mode:** This is the most common possibility to insert a new task in the plan. The task has to be provided with a certain number of preconditions (in terms of expected events), which satisfaction can be specified either as *mandatory* or *optional*: in the first case, the satisfiability itself should be permanently satisfied, *i.e.* if the precondition happens not to be satisfiable anymore, then the task is aborted. On the contrary, an *optional* precondition is considered as satisfied (and hence removed from the task's list of preconditions) if it is actually satisfied *or* if it happens that its own satisfiability becomes **unsatisfiable**. In this case, the task is not aborted. Figure 4 illustrates these precondition mechanisms.
  - **Very Urgent Task (VUT) mode:** this mode is a way to trigger a priority task, preventing any incompatible task to be executed during this time: the list of incompatible tasks to prevent should be provided as parameters of the task insertion. If an incompatible task is already running, it is interrupted. Otherwise, if an incompatible task is scheduled, then it can be either canceled (and de-scheduled) or only delayed (its preconditions are updated taking into account the task being inserted in VUT mode). The expected effect on scheduled incompatible tasks should be specified as well in the parameters of the task being inserted.
  - **DEPendant (DEP) mode:** it is a shortcut to insert a task with as many preconditions as tasks currently scheduled: each precondition is satisfied when the corresponding task triggers its "end of task" event.

Moreover, these are *mandatory* preconditions (*i.e.* as defined in the SEQ insertion mode).

– **Non Urgent Task (NUT) mode:** it is also a shortcut to insert a task, setting as many preconditions as tasks currently scheduled: each precondition is satisfied when the corresponding task triggers its "end of task" event. However, contrary to the DEP mode, these are *optional* preconditions (*i.e.* as defined in the SEQ insertion mode).

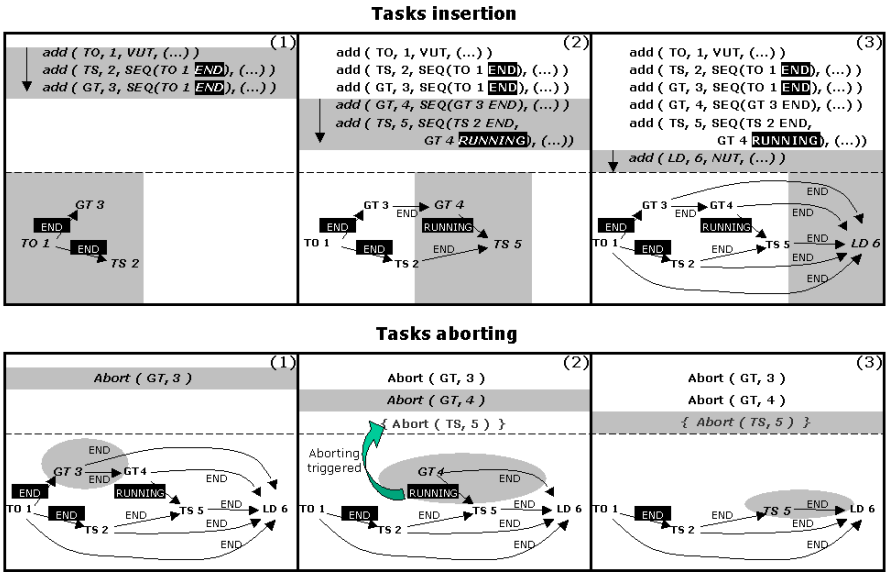


Fig. 4. Top: Examples of tasks insertion and illustration of the corresponding preconditions dependencies. (1): a VUT task and SEQ tasks with single mandatory precondition. (2): SEQ tasks with both mandatory and optional preconditions. (3): NUT task. **Bottom:** Examples of tasks aborting (1) and illustration of aborting propagation to dependent tasks having mandatory preconditions (2) and (3)

- **dynamic tasks aborting:** this mechanism enables the possibility to request tasks abortion in the current plan. If the task is already running, then the abortion of the task is an interruption. If the task is not yet running, then the abortion is a cancellation (the task is de-scheduled). The abortion triggers a propagation mechanism, that checks which of the scheduled tasks depend on the aborted task (*i.e.* the tasks having a precondition expecting an event from the aborted task, like a "end-of-execution" event): if the dependence is a *mandatory* precondition, then this task is also aborted, and so on. If the dependence is an *optional* precondition, then the dependence is removed as if the precondition was satisfied, and

the corresponding task is not aborted.

The level 3 of decisional autonomy introduces an additional mechanism intended to enable autonomous synchronizations from different robots MLEs. A synchronization can be requested to a given MLE as a particular task, that produces events (start, running, end...) in the same way as usual tasks do. It is also possible to insert a synchronization task with particular insertion modes as defined previously. Two "roles" are specified as parameters of a synchronization task: sender ($\mathcal{S}$), and receiver ($\mathcal{R}$): $\mathcal{S}$ and $\mathcal{R}$ are the set of robots considered respectively as senders and receivers of the synchronization. When a synchronization task is processed, the MLE checks whether its own ID is noticed in the $\mathcal{S}$ or $\mathcal{R}$ sets. Three situations may occur:

- ID $\in \mathcal{S}$ (only): the MLE has to send a synchronization signal to all robots which ID belongs to set $\mathcal{R}$. This signal contains the synchronization task's ID, and also this robot's ID. From this robot point of view, the task is considered achieved.
- ID $\in \mathcal{R}$ (only): the robot expects to receive synchronization signals from all robots which IDs belong to the set $\mathcal{S}$. From the point of view of this robot, the synchronization task is considered achieved as soon as all signals are received.
- ID $\in \mathcal{S}$ and ID $\in \mathcal{R}$: the robot should both send its own synchronization signal then wait for signals from all other robots specified in the set $\mathcal{S}$. The synchronization task is considered achieved as soon as all signals are received. If $\mathcal{S}$=$\mathcal{R}$, then the synchronization is a general "rendez-vous" between all robots.

Figure 5 illustrates this synchronization mechanisms.

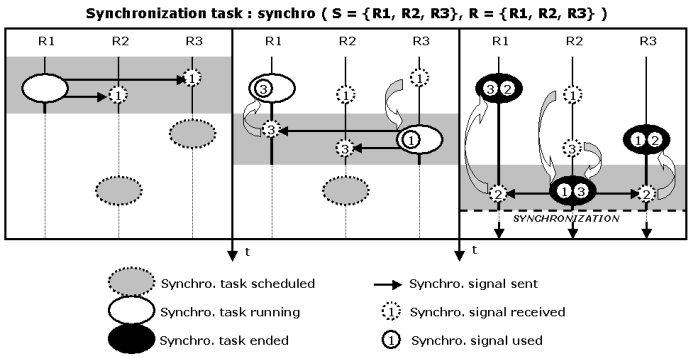


Fig. 5. Illustration of a synchronization task with 3 robots, in the case of a general "rendez-vous"

3.3 Illustration

These various mechanisms have been instanciated within the COMETS project[24], and exploited in a scenario that implied a fire detection, confirmation and monitoring task, plus a mapping task. A video depicting the various phases of the scenario can be seen at *www.laas.fr/˜ simon/eden/gallery/videos.php*.

4 Multi-UAV distributed mission-planning

4.1 General considerations related to the planning scheme

The symbolic planner we use is based on the Shop 2 HTN planner [23], exploiting a hierarchical definition of the planning domain. According to this paradigm, high level *methods* are decomposed into lower level tasks (either other methods or operators) when methods' preconditions are satisfied, until the planner reaches primitive tasks (*operators*).

We introduce time thanks to a particular encoding of the domain based on *Multi-Timeline Preprocessing* (MTL), according to [23]. This scheme enables to express durative and concurrent actions, which is very relevant in robot's tasks planning.

Moreover, we allow, for every task, the possibility to deal with temporal constraints: these time constraints are related to wishes or requirements, expressed in missions requests. Four possible time constraints are enabled in this way: *start before, start after, end before, end after*. When a method generates sub-tasks during its decomposition, these sub-tasks inherit the time constraints.

We distinguish two kinds of operators: *actual operators* (AO), corresponding to explicit tasks in the generated plan, and *convenience operators* (CO), manipulating intermediary data, but not directly dealing with actual robot's tasks.

AOs have the following properties:

- An unique ID (generated during the planning process)
- A dependence list: dependencies dealing with other (previous) operators. This list built using the MTL properties is used when the MLE receives a plan to execute: the dependencies are then turned into preconditions.
- A relative starting time: a time interval where the task's starting should be triggered.
- A duration: provided by the *specialized refiners*.
- Time constraints, inherited from higher level methods decomposition, during planning.
- Some parameters, according to the operation's type.

These AOs mainly match the elementary tasks defined previously (e.g takeoff, gotoXYZ, etc.). AOs may also match highest level tasks which can not be

refined in the only UAV's context : such tasks require multi-UAV refinements, which occur in a second step, trough the *interactions manager*. The duration of such a *Joint Task* (JT) is not necessarily relevant during plan building, since it may depend on the task refinement issue in the multi-UAV context: in this case, the duration is let "unknown" for this task.

On the other hand, the COs are related to intermediary operations, such as calling the specialized refiners during planning. Applying such a CO operator is required before applying any AO operator, since it provides a way to link symbolic knowledge with actual models of the world: environment, UAVs, communications, etc.

4.2 Exploiting the specialized refiners during the planning process

```
(:method (general-gotoxyz ?destloc ?time-constraints)
 ; preconditions
     ((uavloc flying ?startloc) (not (eval (eql '?startloc '?destloc))))
 ; subtasks
     (:ordered (!compute-gotoxyz ?startloc ?destloc)
               (!task-gotoxyz ?id ?dependences ?startloc ?destloc
                              ?waypoints ?start ?duration ?time-constraints)))
)
```

Fig. 6. A Shop method for the generation of a "gotoXYZ" primitive

```
(:operator (!compute-gotoxyz ?startloc ?destloc)
 ; preconditions
     ((assign ?result (compute-data 'gotoxyz (list '?startloc '?destloc))))
 ; delete list
     ()
 ; add list
     ((eval-ok gotoxyz ?result))
 ; cost
     0
)
```

Fig. 7. CO example: calls the specialized refiners features

Figures 6, 7, 8, illustrate a "gotoXYZ" method (fig. 6) giving rise first to the computation (CO, fig. 7) of data related to "gotoXYZ" task, then applying the primitive "gotoXYZ" task (AO, fig. 8). The "compute-gotoXYZ" operator sends a request to the specialized refiners for the refinement of the "gotoXYZ" task, taking into account initial location and destination location, and the returned result is added in the current planning state (through the logical

```
(:operator (!task-gotoxyz ?currentid ?dependences ?startloc ?destloc
            ?waypoints ?start ?duration ?time-constraints)
  ; preconditions
    (
      (eval-ok gotoxyz ?pre-computed-data) ; (1)
      (assign ?duration (get-duration '?pre-computed-data)) ; (2)
      (assign ?waypoints(get-waypoints '?pre-computed-data)) ; (3)
      ...
  ; delete list
      ...
  ; add list
      ...
  ; cost
    (get-cost '?pre-computed-data)) ; (4)
```

Fig. 8. AO example: the "gotoXYZ" operator

atom "eval-ok...", in the operator's "add list" field). Then the "gotoXYZ" operator exploits the corresponding result (line (1) on fig. 8). Finally, the result is parsed into the different relevant data, e.g. duration, waypoints and costs associated to the "gotoXYZ" operation application (resp. lines (2), (3) and (4) on fig. 8).

Figure 9 illustrates an instance of "gotoXYZ" task, as it appears in a final plan.

```
(TASKREQ
   TASK-GOTOXYZ 13
   (DEPENDENCES ((ENDED 12) (ENDED 11)))
   (PARAMS
      (WAYPOINTS ((WP 100.000000 -140.000000 100.000000 0 0 0 0 -1)
          (WP 100.000000 -130.000000 90.000000 14 1 -1 1 -1)
                              ⋮
          (WP 120.000000 -10.000000 50.000000 164 13 -1 20 -1)))
      (START-TIME 240 352.000000)
      (DURATION 88 113.000000)
      (TIME-CONSTRAINTS NIL NIL NIL NIL)
)
```

Fig. 9. "GotoXYZ" task, ready to be executed

Actually, the specialized refiners have the means to process data for much more complex tasks, such as tasks requiring both refinements for perceptions and path planning (e.g. TSP with planned perceptions, see section 4.3).

Exploiting resulting plans - multi-UAV coordination issues

Only the AOs are notified in the final plan. Such a plan is ready to be executed **iff** it does not contain any task requiring coordination with other UAVs, i.e. JTs. However, if the plan contains JTs, the plan coordination is performed in a second step, through the *interaction manager*.

The interaction manager provides the means to coordinate UAVs activities, relying on distributed negotiation mechanisms. All the tasks requiring multi-UAV interactions (simple synchronization or more complex JTs) are processed in the interactions manager, so that the joint operations can be coordinated, for each involved UAV, in terms of space and time.

Detail related to the interactions manager is not provided here, since still ongoing work. Mainly three issues are tackled:

- Temporal coordination: achieved relying on UAVs synchronizations. We defined and implemented a scheme to enable incremental negotiations related to possible time intervals synchronization. As a result, a group of UAVs acknowledge a common time interval in which the synchronization should occur.
- Spatial coordination: we consider interactions models, to reason about the interactions requirements within the JTs. Afterward, during plan execution, collision avoidance can be safely achieved applying a Plan Merging Protocol [22] on the planned trajectories of UAVs.
- Tasks re-allocations: this issue consist in enhancing the global activity of the UAVs, allowing them to re-distribute some tasks, when relevant. For each UAV, the relevance should be assessed w.r.t. the current tasks costs / utility in the current plan (see section 5).

During coordination, the interaction manager may as well request computations / refinements related to the environment and UAVs models, i.e. relying on the *specialized refiners*.

As a result of these coordination processes, a coordinated, ready-to-be-executed (but not necessarily definite) sequence of tasks is provided and inserted in the current MLE's plan.

4.3 The specialized refiners tool-box: overview

The specialized refiners provide a wide set of features to support tasks decompositions and refinements during planning and coordination. They rely on different models (environment, UAV, etc) regularly updated during the UAV's activity, and offer (through a common interface) a set of services related to paths generation, perception planning and communication constraints satisfaction checking. These different processes are performed in a timely manner, so that the symbolic planner may use them in a transparent way during plan building.

The main point here it to provide the planner and the interaction manager with information that will allow them to estimate the ability of the robot to perform a given task in a given (dynamic) context, to compute the various costs and to weight the consequences of inserting a given task into the current robot plan. Hence such information should be sufficiently realistic and produced in an efficient way in order to be used on-line. Indeed, the overall process is incremental and is subject to frequent revisions.

Models

The environment model developed provides two kinds of information: ground surface data and airspace data. The ground model is a 2D array composed of square cells whose attributes are related to fire, mapping and alarm monitoring. A burning factor representing the burning risk is associated to each cell. The airspace model is represented by a 3D array of voxels, and gives relevant information for trajectories and perceptions planning. It indicates whether a voxel is free (and safe) for flying or not. Moreover, considering the communication model (described hereafter), we are aware of the voxel's communication coverage (regarding a control center for instance). Potential waypoints for trajectories planning are nodes located at the center of voxels' facets. A single voxel has 6 nodes and shares each of them with a neighbor. Nodes are connected by edges. Each edge is labeled with the cost for the UAV to move from one node to another adjacent node.

We also use a generic UAV model that provides information concerning flight capabilities and available resources. It is mainly used for flight time estimation purposes; it also gives information about possible orientation for perception devices. The perception model contains technical characteristics related to perception devices (e.g. expected coverage...), and informs about the availability of the sensors. Finally, We have also implemented a (quite simple) communication model that allows to estimate the ability to communicate between two entities. With omni-directional antennas and no solid obstacle between the sender and the receiver, the following conditions must be satisfied:

$$\frac{power_{sender}}{4\pi \left(distance_{sender/receiver}\right)^2} > sensibility_{receiver} \tag{1}$$

According to these models, various "services" can be provided: the next section provides algorithmic details related to these features.

Algorithms

Simple path planning

Path planning is performed here, in a simple way (A* based), in order to compute a path in a discretized 3D environment (section 4.3). The planner takes into account obstacles (hilly ground) and no-flying zones (Figure 10).

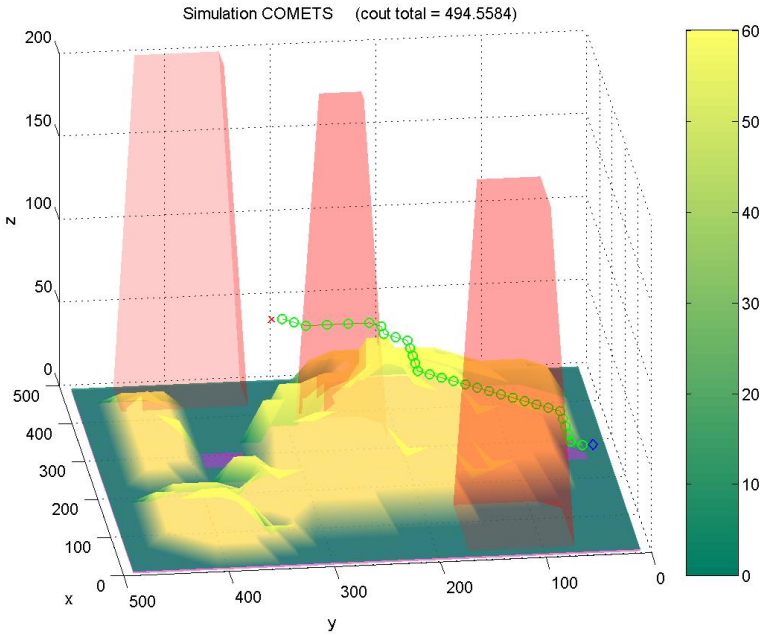


Fig. 10. Simulation for path finding

This planner is used to compute all possible trajectories between all potential robot's mission objectives. We use an extension of this scheme to find the shortest path between several points (i.e. Traveling Salesman Problem (TSP)). This is used by the planner and the interaction manager to compute the best way to insert a new task and to estimate the cost of its insertion in the robot plan.

The TSP is approximated here with a simple stochastic algorithm, using two operations: insertion and permutation. At each iteration, a local *minimum* is found from an initial random solution. After a given number of iterations the best result is kept as the global solution. The number of iterations has been chosen experimentally, as a trade-off between the quality of the solution and the computation time. The algorithm gives quite homogeneous results (as far as the number of points is not less than 30, which is largely enough for a complete UAV mission).

Mapping

The goal of the mapping task is to cover a whole given area in the shortest time. In this problem, we try to minimize the number of turns, according to [25], where authors introduce an efficient method of area decomposition for multi-UAV contexts and a relevant way to apply sweeping patterns. Turning

is considered as critical because an UAV may slow down as its direction is changing. Moreover, trajectories are more difficult to follow in turns, hence perception can't be achieved as efficiently as in straight lines. The principle of the mapping algorithm is to select a favored direction (along the longest straight line inside the area), and then to apply a sweeping pattern considering this direction, as shown on fig. 11. We assume that areas are (or can be divided) into convex polygons.

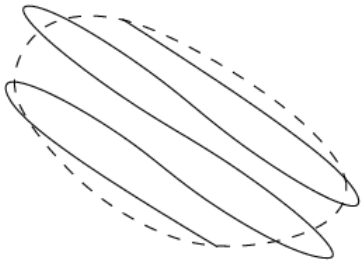


Fig. 11. Example of sweeping pattern, for mapping/coverage applications

Detection

This activity requires the UAV to fly over an area during a given amount of time, trying to minimize the time between two flights over a given ground cell. Moreover, different priorities can be associated to the cells. For instance in the context of the COMETS project, dry vegetation is more likely to burn than water pools: detection activity should be performed with respect to terrain's burnability.

We propose an algorithm based on potential fields. Each cell of the ground is associated to a point of the field, initiated to its maximum, and decreasing with the time according to the equation :

$$P = e^{-r \cdot \Delta T}, \tag{2}$$

where P is the potential of the considered point, $r \in [0, 1]$ is the risk factor and ΔT is the time since the last visit. Perception is not limited to a single cell, depending on the perception device's aperture and flying altitude. At each step of time, ΔT is incremented according to the factor

$$1 - e^{-\left(\frac{(x-x_u)^2+(y-y_u)^2}{\sigma^2}\right)}, \tag{3}$$

where (x, y) is the location of a cell, (x_u, y_u) is the location of the UAV and σ is a parameter representing the sensor's aperture (and so its influence on adjacent cells). After each increment, the value ΔT of the cell located under the UAV is set to zero, corresponding to the maximum of P. The next move

follows the steepest gradient in the potential field. Fig. 12 shows the state of the potential field during a simulation.

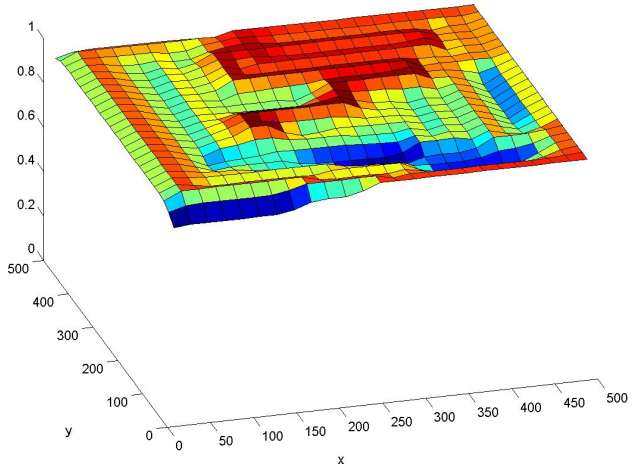


Fig. 12. Example of potential field for detection applications

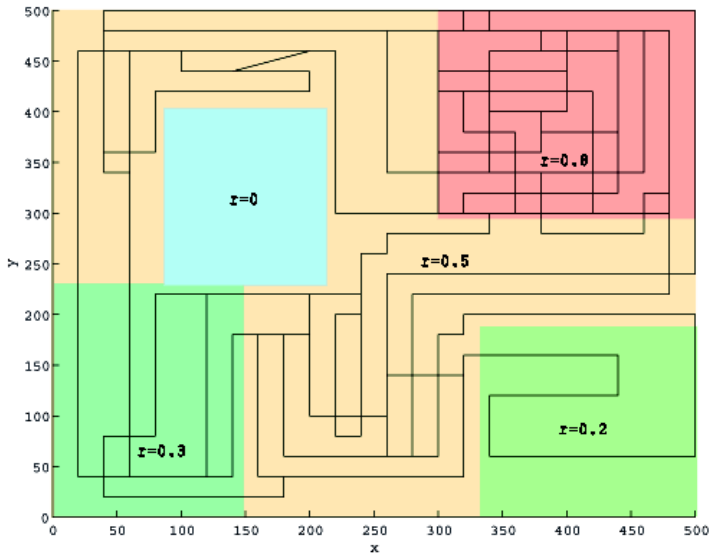


Fig. 13. Simulation results for detection over an heterogeneous area

Fig. 13 provides an example of path followed by an UAV over an area with heterogeneous parts : high risk parts clearly appear with a large number of flights, whereas low risk areas are rarely visited. Even for very low risk area, the potential slowly decreases until reaching a lower value than high risk areas, and hence also attracts the UAV after a given lapse of time. As a consequence, except if r is equal to zero, every cells are explored at least once, after a long enough time.

Constraining basic requests

When communications need to be maintained between an UAV and another entity (e.g. other UAV or control center...) during a flight, we have to check if related voxels are in communication range with the entities. If it is not the case, a "filter" can be enabled in order to prevent the path planner to build paths across these voxels (increasing the traversability cost of these voxels, for instance). Other types of constraints can be applied through basic requests, such as a *preliminary path computation*: the refiner should take into account a preliminary path, before to perform the requested refinement. Indeed, the initial location can have an influence on the refinement issue.

4.4 Illustration

This section illustrates the developed planning scheme in the context of scenario that involves three UAVs.

Mission and scenario

Mission

The general mission's goal is to perform fire detection and fire monitoring over a given area A. The initial task allocation is performed by a human operator. Fire alarm detection should be performed by one UAV over A. Every located alarm should be confirmed by another UAV. If an alarm is confirmed, then 2 UAVs should perform coordinated monitoring around the fire.

Scenario

Three UAVs are introduced in this scenario: one blimp (K), not very maneuverable but well adapted to high altitude flights, and two copters (H and M), much more maneuverable, having hovering capabilities, well adapted to low altitude flight.

K is requested to perform detection over A. After a certain amount of time, a first fire alarm is raised over the location L1, then a second fire alarm is raised over the location L2: H is requested to make perceptions around L1, and M should make perceptions around L2. In L1, the alarm is infirmed (false alarm). In L2, the alarm is confirmed: H is requested to perform coordinated

perceptions with M around L2, for monitoring purpose (requires a synchronization of the monitoring). During this time, K keeps on performing fire detection around L1 and L2. The monitoring activities performed by H and M should go on until K's detection activity is ended. After a certain amount of time, K stops its detection activity: a synchronization signal is sent to H and M. All the UAVs come back to the base station.

Running the scenario

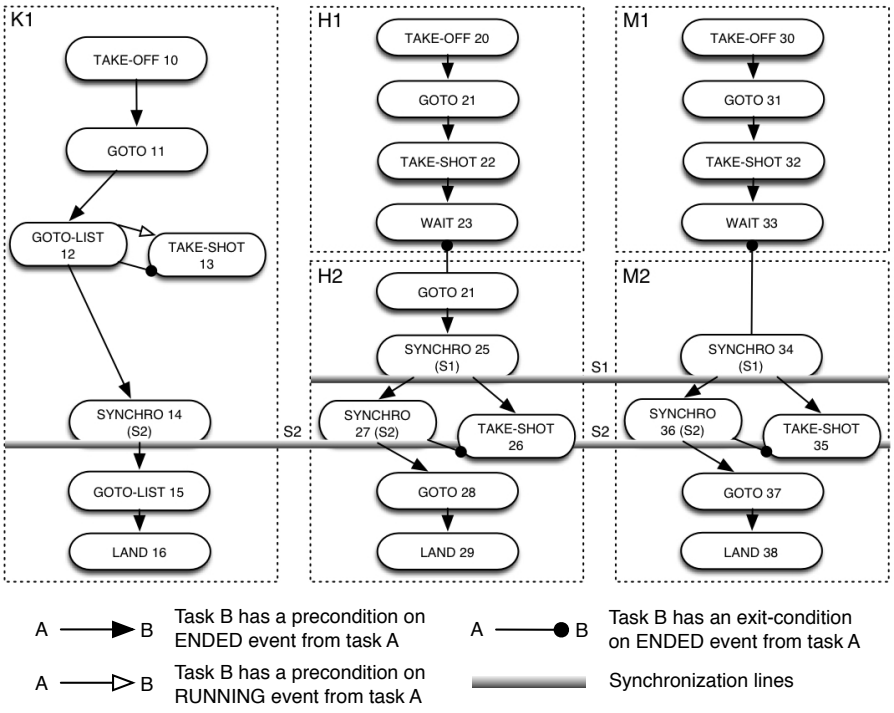


Fig. 14. Example of scenario involving 3 UAVs: K,H and M's plans

Requests hereafter deal with high level Shop methods: once requested to Shop, they are decomposed into refined elementary tasks (resulting UAVs' refined plans are illustrated on fig. 14), exploiting the specialized refiners abilities. The sweeping pattern for fire detection is computed by the specialized refiners, as well as the most fitted perception locations close to L1 and L2 (for H and M), maximizing the perception utility (figure 15 depicts a simulated instance of this scenario).

K blimp's mission

- K should perform detection over A during 15 minutes;
- THEN K should send sync.signal (S1) to H and M.
- THEN K should come back to the base station.

On fig. 14, task 11 is a "goto" task leading to area A. Task 12 is a "goto-list" task associated to the detection pattern computed by the specialized refiners. As task 12 is running, the perceptions are simultaneously triggered (task 13). Then once the synchronization is achieved, the "goto" task 15 makes K come back to the base station.

H copter's mission (part 1: H1)

L1 alarm raised (trough K's perceptions): should be confirmed by H.

- H should make perceptions in L1 during 1 minute.
- THEN H should wait for further orders in secure mode.

Task 21 (fig. 14) is a "goto" task leading to L1.

M copter's mission (part 1: M1)

L2 alarm raised (trough K's perception): should be confirmed by M.

- M should perform perceptions in L2 during 1 minute.
- THEN M should wait for further orders in secure mode.

Task 31 (fig. 14) is a "goto" task leading to L2.

M copter's mission (part 2: M2)

L2 confirmed (M's perception): should perform coordinated monitoring.

- M should perform monitoring activity of L2 with H until receiving synchronization signal from K.
- THEN M should come back to the base station.

On fig. 14, task 34 is the synchronization with H for monitoring (task 35). Task 36 is the synchronization with K, which achievement stands as exit condition for task 35. Then task 37 is the "goto" task back to the base station.

H copter's mission (part 2: H2)

L1 alarm is wrong, and L2 is confirmed (trough M's perception data processing): should perform coordinated monitoring.

- H should perform monitoring activity of L2 with M until receiving synchronization signal from K
- THEN M should come back to the base station.

On fig. 14, task 24 is a "goto" task leading to L2. Then task 25 is the synchronization with M for monitoring (task 26). Task 27 is the synchronization with K, which achievement stands as exit condition for task 26. Then task 28 is the "goto" task back to the base station.

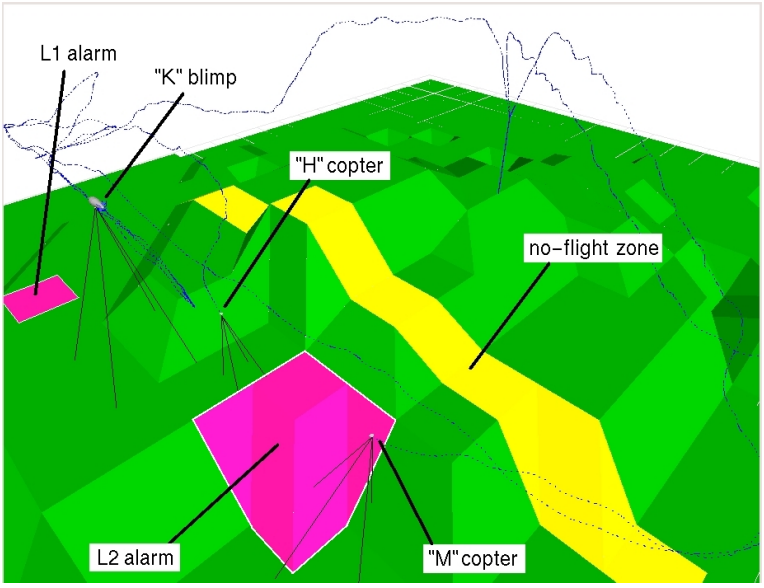


Fig. 15. Level 4 features in simulation: coordinated monitoring over L2

5 Distributed task allocation

Problem statement

The focus in this section is put on the distributed task allocation issue, an ability the level 5 robots must be endowed with. Given a system of several robots, and given a mission defined as a a partially ordered set of tasks, we want the robots to allocate all the tasks to each other and build their plans accordingly in order to complete the mission. They should also be able to dynamically modify the allocation, and consequently their plans, to adapt to changes in the environment or to new requests issued by the operator. The system must also satisfy constraints on energy resources and communication ranges, that are both limited. Also, all the robots must ensure sufficient energy to go back to their starting point when their tasks are achieved.

In the context of an environment surveillance mission, the tasks the system of robots must achieve consist of (i) navigation tasks ($i.e.$ reach a given position), and (ii) perception tasks. The latter tasks can be achieved while the UAV hovers at a given position, or while it follows a predefined path ($e.g$ circling around a given target), but they can also imply the simultaneous presence of several UAVs, hovering or moving according to predefined geometric patterns. Therefore, two distinct problems must be solved:

1. allocation and planning of navigation tasks,
2. constraints on tasks schedules which enable the system to deal with complex tasks that need synchronization of robots activities.

The first problem is a multi Traveling Salesmen Problem (often referred as *m-TSP*). The second problem is an extension of the previous one, in which constraints on the execution dates of the tasks must be satisfied: it is possible to stipulate for a task the date when it must be started with respect to the start date of another task.

This section introduces a task allocation scheme based on *Contract-Net*, that is innovative in two ways :

- For the first problem, we aim at minimizing a global criteria (the longest trip) while *Contract-Net* only takes into account local data. We introduce in *Contract-Net* a global parameter which helps the optimization the criteria. It also helps to control the auctions generation in the system.
- For the second problem, the robots must share up-to-date data that describe the constraints on the tasks and need to plan the tasks that are linked together accordingly, without disrupting the bidding process of *Contract-Net*. This will be done with the temporary assignment of master/slave role to the robots depending on the tasks that have been allocated. The challenge lies in avoiding the use of a centralized planning algorithm.

Related work

Rabideau *et al.* made a comparison of several methods for tasks allocation in [26]. They emphasize algorithms with three degrees of distribution, the most distributed one is based on the *Contract-Net* protocol. In [27] Bellingham *et al.* successfully implement in simulation an algorithm for the *optimal fleet coordination problem*, their algorithm does not address the problem of synchronization between tasks and can be classified in level 2 from [26]. Dias and Stenz also studied various approaches to the task allocation problem with multiple robots [28] and came to the point that distributed algorithms based on *Contract-Net* suit the needs. Note that a number of distributed schemes for task allocation in multi-robot domains have been proposed in the literature. One of the first is [29]. ALLIANCE [30] is a distributed behavior-based architecture, which uses motivations that enable/inhibit behaviors, resulting in tasks (re)allocation.

Contract-Net has been introduced by Smith in [31] and further developed by Sandholm [32]. Since 1999 *Contract-Net* have been widely used in multi-robot applications [33, 34, 35]. Stenz and Dias work on an architecture called *TraderBots* in which *leaders* can optimize the plan of several other robots [36, 37], Mataric explored various strategies for the *Contract-Net* protocol in [38]. Several studies dealing with concrete mission such as buildings or planetary exploration [39, 40, 41] or emergency handling [38], have shown the feasibility and the performance of the *Contract-Net* architecture in real world situations.

To our knowledge, on the problem of allocation and planning of non-independent tasks in a distributed multi-robot system, only one paper from

Kalra and Stentz [42] presents preliminary results on the *sweeping perimeter problem*. In this work, the temporal window taken into account is very small, the coordination is explicit between a limited number of robots (one robot and his two neighbors), and the market-based approach is not fully exploited since the auctions imply three agents only.

The next section introduces an *equity coefficient* that is used in the bids evaluation and to control the auction generation process within *Contract-Net*. Quantitative simulation results obtained on the m-TSP problem illustrate the improvements brought by the consideration of this coefficient with respect to a plain *Contract-Net* approach. Section 5.2 deals with the introduction of time constrained tasks. It shows that the introduction of simple execution date constraints can help to cope with cooperative tasks, that are either requested by an operator or automatically generated within the system, to establish communication relays for instance.

5.1 Contract-Net with equity

In the classic market-based approach, each agent (for us robots) can make a public auction for one of its tasks, and then the other robots can bid on that task using a given cost function. The winner of the bidding process gets the task and must insert it in its plan. In order to drive the process toward an optimal solution, one agent can sell a task only if the bidden cost for the execution of that task is at least less than a certain amount of its own execution cost (generally 10% less). The cost function we will use here is simple and is calculated from the distance the robot will travel.

Equity factor

The aim is to obtain an allocation that minimizes the length of the longest trip, which also can be seen as minimizing the duration of the mission. Our idea is to address this global optimization problem by considering two aspects: first, *Contract-Net* is used to assign tasks to the robots at a low cost so as to keep the total distance traveled by the team of robots not too far from optimality, and second, equity is enforced between the robots so as to really distribute the tasks among them and obtain a mission which is as short as possible.

For this purpose, we introduce a measure of equity called *equity coefficient* (C_{eq}). Each robot can compute its own workload (wl) using a cost function: the workload is the cost of the whole plan of the robot. The robots broadcast the value of their workload to the others and each one can compute its C_{eq}. For the robot A the formula is :

$$C_{eq}^{A} = \frac{wl(A) - \overline{wl}}{\overline{wl}} \tag{4}$$

Where $\overline{wl}$ is the mean of $wl(.)$ over the robots for which A knows the workload. Indeed, since we consider limited communication range, A may have only a partial knowledge of the workloads. The meaning of this coefficient is :

- $C_{eq}^{A} < 0$: robot A has a too small plan with respect to the other robots.
- $C_{eq}^{A} > 0$: robot A is overloaded with respect to the other robots.
- $C_{eq}^{A} > C_{eq}^{B}$: robot A has more work than robot B.

Equity factor and task evaluation

In *Contract-Net* a task is allocated to the robot which can insert it in its plan for the lowest cost; also the robot should not be too overloaded. For that the evaluation the robot makes for a task is modified by taking into account its C_{eq}. The utility the robot A computes for the task T_1 ($ut^A(T_1)$) is corrected in $ut'^A(T_1)$ by :

$$ut'^A(T_1) = ut^A(T_1) - C_{eq}^{A} \times |ut^A(T_1)|$$

This correction is applied to the utility computed by both the auctioneer and the bidder. By this mean *Contract-Net* is influenced the way we want :

- A robot with a high workload will more easily reallocate its tasks and will get new tasks with more difficulty because its utility for the tasks is lowered.
- On the contrary, a robot with a low workload will be more easily allocated new tasks but will give up its own tasks with more difficulty because its utility for the tasks is increased.

Control of the auctions generation

The problem here is we do not want several auctions being launched at the same time. Basically, the *Contract-Net protocol* does not provide any details when the agents of the system can start an auction, and other papers do not emphasize this point either. Our need is to keep the system entirely distributed, so we do not want an authority which would give the right to the robots in turns, and we want to keep the system dynamic so we do not want to give to each agent a static list which would define the turns for the auctions.

Our solution is inspired by the *token-ring* networks in which a token passes from one computer to another to give them the authorization to send their data over the network. Here the token allows the robot to make an auction.

Token circulation

The robot that has got the token is the auction leader. If another robot is willing to make an auction, it can ask for the token to the current auction leader. It sends its request along with its C_{eq}. The owner of the token collects

all the requests, it is also allowed to request for the token. It then randomly chooses the next owner of the token, using a random distribution based on the collected C_{eq} (the more a robot is overloaded, the higher chance it has to get the token). This is done to help overloaded robots to reallocate their tasks.

Token creation

When a robot wants to make an auction, but nobody has the token, it then creates a token and uses itself to make an auction, the process is started spontaneously ! Because of communication delay, it may happen that several robots create a token at the same time, this is why we specify the following behavior :

- If a robot that is not currently an auctioneer receives several auctions at the same time, it then bids on the auction which has the higher priority *ie* the higher C_{eq} (the auctioneer gives its C_{eq} along with the auction). The other auctions are ignored.
- If a robot that is currently an auctioneer receives other auctions, it keeps on its own auction only if it has the highest C_{eq}, else it cancels its auction and can bid on the auction with the highest priority.

Results

A typical mission allocation and execution goes this way: (1) A set of tasks is given to the system (either directly by an operator, or issued from a decomposition process). (2) The base station is also a *Contract-Net* agent except that it has a high priority (an artificially high equity factor) and will never make a bid. The base starts making auctions with the tasks of the mission and goes on until all tasks are allocated to the robots. (3) The base does not keep the token any more and the robots can start making auctions. (4) The process stops when none of the robots asks for the token. A robot stops making auction when it has already auctioned all its tasks and no reallocation has occurred. If not, the robot auctions again all its tasks. This stop criterion is quite different from what has been done until now (usually a fixed number of auctions turns). (5) The mission starts being executed by the robots. The auction process starts again when new tasks are requested by the operator, or when a robot fails to achieve its plan.

n the tests, we focus on steps 2 to 4[2]. We based all our tests on the same mission: 50 points picked up in the environment have to be visited by a team of 4 robots. The points have been uniformly randomly generated once in this environment. In order to show the interest of our *equity coefficient*, we run scenarios with the coefficient disabled (we give it a fixed value so as to mimic a plain *Contract-Net protocol*). Another important point is how the robots

[2] The results were obtained with a multi-robot simulation developed in Java.

are firstly distributed in the environment. If they are scattered (the usual situation when *Contract-Net* is used) each robot is implicitly attributed a different area, the area surrounding its initial position, because of the cost function which is based on traveled distance. If they are initially grouped around a same point (which is mostly the case in operational situation), the problem is more difficult.

The results presented in table 16 show that the solution obtained with the equity coefficient is improved by a factor of 2.4 over the standard *Contract-Net*, if the robots are scattered, the improvement factor drops to 1.3. This is due to the fact that the solution found by the standard protocol is already a good one. The interest of our method is that it works well even if the initial situation is not favorable. On the other side the allocation process is about 20% longer (more auctions are done) with the equity coefficient enabled.

scenario	$\bar{l}$	$\sigma(l)$	$\min(l)$	$\max(l)$	n
grouped/no equity	5248	674	3035	6150	133
grouped/equity	2195	264	1844	2954	156
scattered/no equity	2481	481	1724	4631	133
scattered/equity	1895	162	1581	2343	160

Fig. 16. This table summarizes the statistical results over 100 runs of the simulation of four scenarios, considering a grouped or scattered start, and with or without the use of the equity coefficient. l is the length of the maximal tour, and n is the number of auctions of the allocation process.

5.2 Time-constrained tasks in a distributed environment

The problem of constrained tasks allocation and planning for a system of multiple robots is commonly addressed with a centralized planner such as *GRAMMPS* [43]. We sketch here how we deal with simple time constrained tasks in our distributed environment.

Execution around date *d*

This constraint means that we will try to have a given task executed more or less at a given date. This constraint enables the system to deal with constraints on relative date of execution of several tasks expressed numerically: T_1 and T_2 simultaneously or T_1 n seconds before of after T_2.

We choose to put the constraint *execution around date d* on the tasks for several reasons :

- This constraint is *soft*, which means that there is an infinite number of solutions that satisfy it, the distributed allocation algorithm will more easily find a solution, even a bad one, and will not end to dead-lock.

- The quality of satisfaction for such a constraint is easily *measurable*, and then we are able to take into consideration this measure when we evaluate its utility for a given robot. The quality of satisfaction for the constraint can be directly included in the bid of our *Contract-Net protocol*.
- The information needed to plan such constrained tasks is very limited and will not overload the communication bandwidth between the robots.

Constrained tasks tree

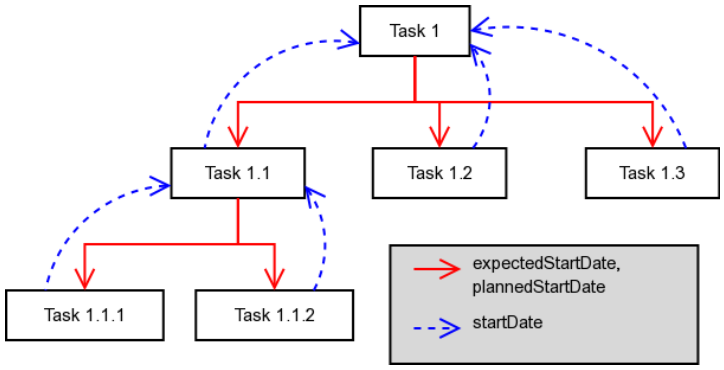


Fig. 17. This tree shows up the hierarchical links between tasks.

The constraint can be used for example to enforce simultaneous execution of two tasks. One task T_1 which is planned for execution at date d_1 puts on task T_2 the constraint *execution around date* d_1. The task T_1 is said to be the mother task, and task T_2 the child task.

Temporally, T_1 is defined by a *startDate* d_1 (the date when it can be executed at the earliest), and a *plannedStartDate* (the date when it will actually be executed). T_2 has the same attributes plus an *expectedStartDate* d_1 (the preferred date for its execution).

The allocation process must allocate T_1 before T_2 (because we need to know d_1 for bidding on T_2), but the tasks can be reallocated later. It is important to note that only T_2 is constrained, T_1 is allocated and firstly planned as usual. After the allocation process, the *master* robot R_A (the one which will be executing T_1) will choose *plannedStartDate* for both T_1 and T_2, the robot R_B (the one which will be executing T_2) is called the *slave* robot. Since the system is dynamic, changes can be made to the plans of R_A and R_B. If it happens, the slave robot only informs its master of the changes in its plan, it sends the new *startDate* for that task and then the master robot computes a new *plannedStartDate* for the execution of the two tasks which is acceptable by both R_A and R_B.

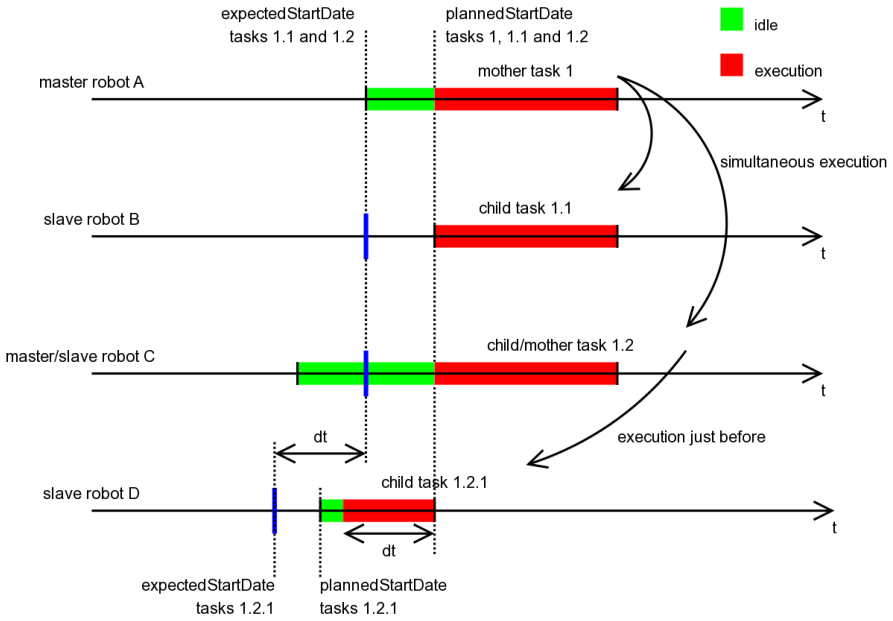


Fig. 18. This is an example of the plans of four robots after the allocation of four constrained tasks.

The relation master/slave between the robots is local in time (only for the execution of the considered tasks), and temporary because the tasks can be reallocated to other robots. So this is quite different from the *TraderBots* architecture [37].

Figure 17 presents a tree of tasks and focus on the data that are exchanged in order to plan the tasks, and figure 18 sketches the allocated tasks from the robot point of view. The synchronization between robots is actually accomplished with the introduction of *idle* periods in the robots plans.

Evaluation of the utility of a task

Now the quality of satisfaction for the constraints which weight on the tasks of the plan is to be taken into account. Previously we computed the cost of a plan with its length; now we add a term for each task which reflects the constraint satisfaction quality. We call this term *deltaDate*, for the constrained task T_i the formula is :

$$deltaDate_i = |startDate_i - expectedStartDate_i| \quad + \sum_j deltaDate_{ij}$$

where $deltaDate_{ij}$ comes from the children tasks T_{ij} of task T_i. These children tasks are either allocated to the same robot or to another one.

The utility of a plan can now be computed by the formula :

$$planUtility = -\left[movingCost + k \times \sum_{task_i \in plan} deltaDate_i\right]$$

The robot bids on a task with the value $(planUtility' - planUtility)$ where $planUtility$ and $planUtility'$ are respectively the utility of the plan *before* and *after* the insertion of the task.

The factor k is here to normalize the sum. In fact we add two quantities $movingCost$ and $deltaDate$ which are not of the same nature. This becomes false if the $movingCost$ is computed with the time needed by the robots to go from one point to another. One can understand k as a scale factor, $k = 0.1$ (it is the typical value we use.) means that we find the periods when the robot is idle 10 times less important than when the robot is active.

Time consistency of the plans

We must ensure that the time constrained tasks are planned correctly to prevent the system from ending into deadlock. Here again we use a very basic planner, not really efficient, but very easy to implement and which clearly maintains consistency of the plans. Each child task is tagged with an $expectedStartDate$, the planner will insert the task into the plan so as to respect the local chronology between children tasks of this plan.

When a modification occurs in the plan, we use a simple but rough process to maintain this local consistency: if two children tasks are not in the chronological order, they are swapped.

Assuming that the plans are incrementally built, the local consistency ensures the global consistency. Indeed, the synchronization is reached by inserting idle periods into the plans of the robots so a robot waits even a long time for synchronization rather than trying to swap tasks.

Implemented tasks

Two new types of task have been implemented in our simulation to illustrate constrained tasks: *watchout* and *com-relay*.

The *watch-out* task consists for the robot to travel around a rectangular area to be monitored and the robot must keep communication with the base. If the communication link between the base and the robot cannot be maintained during the execution of the task, then the robot should generate a *com-relay* task between it and the base which is to be executed by another robot at the same time the watch-out task is executed. The com-relay task can be recursive, which means that several robots can be needed to effectively maintain communication between the base and the robot which will be watching out the area. Figure 19 illustrates these two tasks.

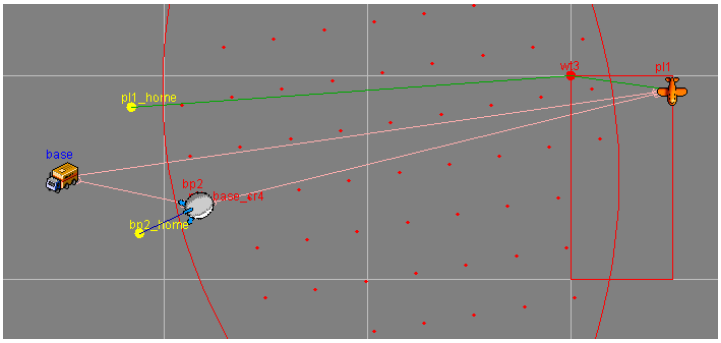


Fig. 19. This screenshot presents a plane *pl1* which is watching out an area and a blimp *bp2* which is a communication relay between *pl1* and the base. The pink lines represent the communication links which are available (the link *pl1*→base is not available). The red circle arcs enclose the area where the blimp can serve as a communication relay, and the dots represent the discrete positions, the planner of the blimp has chosen one of these positions.

Illustrations

Here the results are more qualitative. The simulator shows that even with very simple planning algorithms in a distributed environment, we manage to allocate and plan a mission correctly. Figure 20 presents what we obtain with our simulator on some examples. It illustrates the strategy found by the team: since for the watch-out tasks two robots are needed (one for the given task, and one more for the com-relay), we can see that the four robots are split into two teams of two robots and each team takes care of a part of the environment. When there remains only three robots, the solution is more complex and less structured, but is valid and does not appear to be very sub-optimal.

6 Summary

This chapter provided insights on the definition of the architecture and associated algorithms to allow the deployment of a fleet of heterogeneous UAVs. It introduces five levels of autonomy for an UAV integrated within a multi-robot system, and proposes algorithms to fulfill three main decision-making functionalities: an executive system common for all autonomy levels that handles the coordination and task execution issues, a planning scheme based on a Hierarchical Task Networks planner completed with plan-refiners that consider the actual domain models, and an instance of the contract-net protocol that can handle task allocation for complex multi-UAV missions.

Further developments should however be made towards the overall integration of the various concepts and algorithms. For instance, the planner used

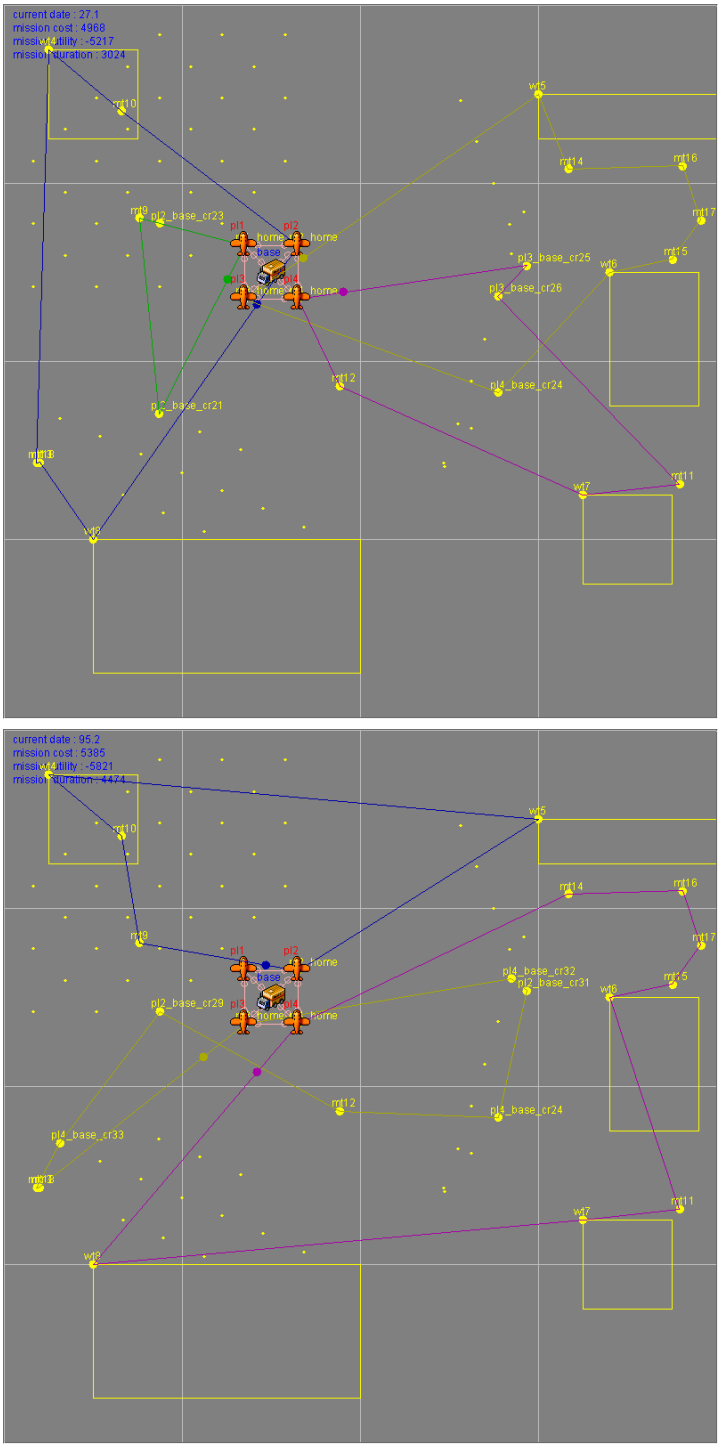


Fig. 20. Top: some watch-out and goto tasks are allocated to the team of robots. Bottom: The plane *pl4* (top-left) has fallen out of order and the tasks have been reallocated to the remaining robots.

in the task allocation process to insert the tasks into the plans of the robots is very simple, and the refiners could advantageously be used instead.

References

1. P. Doherty, G. Granlund, K. Kuchcinski, E. Sandewall, K. Nordberg, E. Skarman, and J. Wiklund, (2000) The witas unmanned aerial vehicle project, in Proc. of the 14th European Conference on Artificial Intelligence, Berlin, Germany, pp. 747–755.
2. S. Sukkarieh, E. Nettleton, J-H. Kim, M. Ridley, A. Goktogan, and H. Durrant-Whyte (2002). The anser project: Multi-uav data fusion, International Journal on Robotics Research, 22:7–8, pp 505–540.
3. S. Sukkarieh, A. Goktokan, J-H. Kim, E. Nettleton, J. Randle, M. Ridley, S. Wishart and H. Durrant-Whyte (2002). Cooperative data fusion and control amongst multiple unihabited air vehicles, 8th International Symposium on Experimental Robotics, Sant'Angelo d'Ischia (Italy).
4. L. Buzogany, M. Pachter and J. D'Azzo (1993). Automated control of aircraft in formation flight, AIAA Guidance, Navigation and Control Conference, Monterey, CA (USA), pp. 1349–1370.
5. C. Schumacher and S.N. Singh (2000). Nonlinear control of multiple UAV in close-coupled formation flight, AIAA Guidance, Navigation and Control Conference, Denver, Co. (USA).
6. T. Balch and R. Arkin (1998). Behavior-based formation control for multirobot teams, IEEE Transactions on Robotics and Automation, 14:6, pp 926–939.
7. F-L. Lian and M. Richard (2002). Real-time trajectory generation for the cooperative path planning of multi-vehicle systems, 41st IEEE Conference on Decision and Control.
8. R.L. Raffard, C. Tomlin and S.P. Boyd (2004). Distributed optimization for cooperative agents: application to formation flight, 43rd IEEE Conference on Decision and Control, Nassau (Bahamas).
9. F. Giulietti, L. Pollini abd M. Innocenti (2000). Autonomous formation flight, Control Systems Magazine, 20:6, pp 34–44.
10. J. Sousa, T. Simsek and P. Varaiya (2004). Task planning and execution for UAV teams, 43rd IEEE Conference on Decision and Control, Nassau (Bahamas).
11. S. Zelinski, T.J. Koo and S. Sastry (2003). Hybrid system design for formations of autonomous vehicles, 42nd IEEE Conference on Decision and Control.
12. R. Vidal, S. Sastry, J. Kim, O. Shakernia, and D. Shim, (2002) The berkeley aerial robot project, in IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Workshop on Aerial Robotics, Lausanne (Switzerland).
13. E. King, M. Alighanbari, Y. Kuwata, and J. How (2004). Coordination and control experiments on a multi-vehicle testbed, IEEE American Control Conference, Boston, Ma. (USA).
14. I. Mazza and A. Ollero (2004). Multiple UAV cooperative searching operation using polygon area decomposition and efficient coverage algorithms, Distributed Autonomous Robots Systems, Toulouse (France).
15. R. Brooks (1986). A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation 2:1, pp 14p-23.

16. R. Arkin (1990). Motor Schema-Base Mobile Robot Navigation. International Journal of Robotics Research.
17. R. Alami, R. Chatila, S. Fleury, M. Ghallab and F. Ingrand (1998). An Architecture for autonomy. International Journal of Robotics Research. Volume 17, pp 315–337.
18. E. Gat (1991). Integrating planning and reacting in a heterogeneous asynchronous architecture for mobile robots. In SIGART Bulletin 2, pp 17–74.
19. R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das (2001). The claraty architecture for robotic autonomy. Proceedings of the 2001 IEEE Aerospace Conference, Big Sky, Mt. (USA).
20. R. Simmons, T. Smith, M. Dias, D. Goldberg, D. Hershberger, A. Stentz and R. Zlot (2002). A Layered Architecture for Coordination of Mobile Robots. Multi-Robot Systems: From Swarms to Intelligent Automata, Proceedings from the 2002 NRL Workshop on Multi-Robot Systems, Kluwer Academic Publishers.
21. L. Parker (1998). ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. In IEEE Transactions on Robotics and Automation, 14:2, pp 220–240.
22. R. Alami, F. Ingrand and S. Qutub (1998). A Scheme for Coordinating Multi-Robot Planning Activities and Plans Execution, European Conference on Artificial Intelligence.
23. D. Nau, T. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman (2003). shop2: an HTN planning system, Artificial Intelligence Research, vol. 20, pp. 379–404.
24. Ollero A. & Al.(2004). Control of Multiple Heterogeneous Unmmanned Aerial Vehicles: Architecture and Perception issues in the COMETS project. IEEE robotics and automation magazine, vol. 12, no. 2, pp. 46–57.
25. I. Mazza and A. Ollero, "Multiple uav cooperative searching operation using polygon area decomposition and efficient coverage algorithms," in *Proc. of DARS'04*, 2004.
26. S. Chien, A. Barrett, T. Estlin, and G. Rabideau, "A comparison of coordinated planning methods for cooperating rovers," in *Proceedings of the Fourth International Conference on Autonomous Agents*, C. Sierra, M. Gini, and J. S. Rosenschein, Eds. Barcelona, Catalonia, Spain: ACM Press, June 2000, pp. 100–101, poster announcement.
27. J. Bellingham, M. Tillerson, A. Richards, and J. P. How, ch. Multi-task allocation and path planning for cooperating UAVs.
28. M. B. Dias and A. T. Stentz, "A market approach to multirobot coordination," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI -TR-01-26, August 2001.
29. H. Asama and K. Ozaki, "Negotiation between multiple mobile robots and an environment manager," in *IEEE Int. Conf. on Robotics and Automation (ICRA'91)*, 1991, pp. 533–5382.
30. L. Parker, "Alliance: An architecture for fault tolerant multirobot cooperation," *IEEE Trans. on Robotics and Automation*, vol. 14, no. 2, pp. 220–239, 1998.
31. R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," in *IEEE Transaction on Computers*, ser. C-29, no. 12, 1980, pp. 1104–1113.
32. T. Sandholm, "An implementation of the contract net protocol based on marginal cost calculations," in *Proceedings of the 11th National Conference*

on Artificial Intelligence. Menlo Park, CA, USA: AAAI Press, July 1993, pp. 256–263.
33. A. T. Stentz and M. B. Dias, "A free market architecture for coordinating multiple robots," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-99-42, December 1999.
34. M. B. Dias and A. T. Stentz, "A free market architecture for distributed control of a multirobot system," in *6th International Conference on Intelligent Autonomous Systems (IAS-6)*, July 2000, pp. 115–122.
35. B. P. Gerkey and M. J. Mataric, "Sold!: Auction methods for multirobot coordination," in *IEEE Transaction on Robotics and Automation*, vol. 18, 2002, pp. 758–768.
36. M. B. Dias and A. T. Stentz, "Enhanced negotiation and opportunistic optimization for market-based multirobot coordination," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI -TR-02-18, August 2002.
37. M. B. Dias and A. T. Stentz, "Traderbots: A market-based approach for resource, role, and task allocation in multirobot coordination," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI -TR-03-19, August 2003.
38. M. J. Mataric', G. S. Sukhatme, and E. Ostergaard, "Multi-robot task allocation in uncertain environments," *Autonomous Robots*, 2003.
39. M. J. Mataric' and G. Sukhatme, "Task-allocation and coordination of multiple robots for planetary exploration," in *10th International Conference on Advanced Robotics*, August 2001, pp. 61–70.
40. R. M. Zlot, A. T. Stentz, M. B. Dias, and S. Thayer, "Multi-robot exploration controlled by a market economy," in *IEEE International Conference on Robotics and Automation*, May 2002.
41. M. B. Dias, D. Goldberg, and A. T. Stentz, "Market-based multirobot coordination for complex space applications," in *The 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, May 2003.
42. N. Kalra and A. T. Stentz, "A market approach to tightly-coupled multi-robot coordination: first results," in *CTA (Collaborative Technology Alliance) robotics program*, 2003. Available at http://www.frc.ri.cmu.edu/ axs/doc/cta03.pdf/
43. B. L. Brumitt and A. Stentz, "GRAMMPS: A generalized mission planner for multiple mobile robots in unstructured environments," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-98)*. Piscataway: IEEE Computer Society, May 16–20 1998, pp. 1564–1571.
44. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem*. New York: John Wiley & Sons, 1985, ch. Empirical analysis of heuristics, Heuristics for the TSP.