Multi-Robot Planning and Execution for an Exploration Mission: a Case Study

Guillaume Infantes, Charles Lesire, Cédric Pralet
ONERA - The French Aerospace Lab, F-31055, Toulouse, France
{Guillaume.Infantes,Charles.Lesire,Cedric.Pralet}@onera.fr

Abstract

This paper presents the first steps of the treatment of a real-world robotic scenario consisting in exploring a large area using several heterogeneous autonomous robots. Addressing this scenario requires combining several components at the planning and execution levels. First, the scenario needs to be well modeled in order for a planning algorithm to come up with a realistic solution. This implies modeling temporal and spatial synchronization of activities between robots, as well as computing the duration of move activities using a precise terrain model. Then, in order to obtain a robust multi-agent executive layer, we need a robust hierarchical plan scheme that helps identifying appropriate plan repairs when, despite the quality of the various models, failures or delays occur. Finally, we need various algorithmic tools in order to obtain flexible plans of good quality.

Motivation

Automatic exploration of large and hazardous areas benefits from the use of multiple robots. Indeed, the use of several robots working in parallel allows the duration of missions to be drastically decreased, which may be useful to deal with crisis situations or with search and rescue missions, in which the response time is a key criterion. The deployed team of robots may be heterogeneous, to take advantage of the skills of different kinds of robots: flying robots can see over buildings and quickly cover large distances; ground robots can accurately map the environment; some robots may be able to enter buildings, overcome obstacles, and even cross flooded areas. Operators often use tools for defining the mission of robots offline, taking into account various operational constraints. Robots must then act autonomously at execution time in order to adapt their behavior to complex, dynamic and uncertain environments, and to perform replanning tasks if needed. Autonomy is especially useful when communications are intermittent or unreliable. In such cases, it is indeed impossible to permanently control each robot from a remote mission center. However, operators often need some feedback at regular time intervals, in order to know how the mission is progressing and to get data collected by robots.

Operational Scenario We consider the problem of exploring an area using heterogeneous autonomous robots, subject to the supervision of human operators . When defining the mission, human operators first define some zones to be observed by the robots. These zones are considered as *observation tasks* that will be allocated to the robots depending on their capabilities (some robots cannot see under trees, or cannot enter buildings). In this work, robots are assumed to be individually able to localize themselves, to plan trajectories, and to perform navigation tasks in the environment, the latter being possibly mapped online using robot sensors.

In the mission considered, human operators need to regularly monitor mission execution. Due to intermittent or unreliable communications between robots, and between robots and the operators, this online monitoring is defined as a time rate at which each robot has to report an execution status for its plan. This execution status may for instance contain the current robot position and the list of zones it has observed. Due to communication and motion constraints, aerial robots, which can move faster, are used to collect other robots data and to communicate these data to the operators. Reporting tasks correspond to temporal and spatial rendezvous, during which robots share information.

Approach Followed The main contribution of this paper lies in the integration of several components for tackling the operational scenario described above. The integrated components are:

- *Multi-agent, temporal and hierarchical planning*: we define a planning algorithm that computes plans of actions for the whole team of robots. Built plans are hierarchical, in the spirit of Hierarchical Task Networks (HTN (Erol, Hendler, and Nau 1994)). Elementary actions in these plans are move, observation and communication actions. These plans enforce temporal constraints coming from the environment model (time needed to move from points to points, or to observe zones) and from the mission requirements (periodic reporting to the human operator). To deal with communications, we propose an offline planning algorithm that includes communication tasks within the initial plans. We use an offline planning approach as a first step to coordinate vehicles which may not be permanently visible from the operation center. The algorithm

used is linked with external libraries that compute for instance durations of moves performed during the exploration, based on a terrain model.

- *Mission execution and repair*: based on embedded reactive architectures, plans are executed on each robot in a distributed way, each robot being in charge of its own tasks. During execution, disturbances may occur, requiring either an adaptation of plans (for instance when a robot is going to be late at a rendezvous), or more global repairs when the plan is no longer feasible. As communication may be unavailable for a global replanning, plans can be locally repaired through communication between a few close teammates (Gateau, Lesire, and Barbier 2013). In this paper, we only consider hard-coded repair rules used in case of failures. More advanced techniques using a deliberative architecture on-board each robot could be considered.

The paper is organized as follows. The next section presents a modeling of the scenario, and the way it is translated into constraints. The general scheme of the execution process (including plan adaptation and repair) is then presented. Afterwards, we present the offline planning algorithm, along with the constraint-based framework the algorithm is based upon. Finally, we present an evaluation of the algorithm on some basic examples, and we show first results on a real-world scenario involving three robots.

This work has been partially supported by the DGA funded Action project (http://action.onera.fr) and the ANCHORS project (http://www.anchors-project. eu), funded by the German Federal Ministry of Education and Research (BMBF) and the French National Research Agency (ANR).

Modeling the Scenario

Static Data

The mission consists in observing a set of n_z zones using n_r robots.

- The team of robots is $R = \{r_i\}_{1 \leq i \leq n_r}$.
- The set of zones is $Z = \{z_i\}_{1 \leq i \leq n_z}$.
- Each robot r_i can reach a set of navigation points $P(r_i)$.
- For each robot r_i, function O_i gives the navigation points from which a zone z_j can be observed by r_i. We have $O_i(z_j) \subseteq P(r_i)$.
- For communications, we consider a function C that gives, for each couple of robots (r_i, r_j) the set of points from which robots r_i and r_j can communicate. We have: $C(r_i, r_j) \subseteq P(r_i) \times P(r_j)$.

Dynamic State Description

At a given time, a robot r_i can be at one of its navigation points $p_j \in P(r_i)$, which is denoted by $at(r_i, p_j)$. A robot r_i can also be navigating between two locations. In normal execution, no action is possible until the robot reaches its destination.

Furthermore, a zone must be observed only once, so we define function $toObs : Z \rightarrow \{\top, \bot\}$ that expresses if a zone z_j is to be observed ($toObs(z_j) = \top$) or if z_j has already been observed ($toObs(z_j) = \bot$).

Task Model

Each robot $r_i \in R$ can perform three elementary tasks:

- $goto(r_i, p_j, p_k)$: navigation from point p_j to point p_k; the preconditions at the beginning of the task are $p_j, p_k \in P(r_i)$, and $at(r_i, p_j)$; as for effects, $at(r_i, p_k)$ becomes true at the end of the task, whereas $at(r_i, p_j)$ becomes false at the beginning of it;
- $obs(r_i, p_j, z_k)$: observation of zone z_k from point p_j; preconditions $p_j \in O_i(z_k)$ and $toObs(z_k) = \top$ must hold at the beginning of the task; condition $at(r_i, p_j)$ must also hold at the beginning and during the task; as an immediate effect, we have $toObs(z_k) = \bot$;
- $com(r_i, r_j, p_k, p_l)$: communication between r_i and r_j, located at positions p_k and p_l respectively; preconditions are $p_k \in P(r_i)$, $p_l \in P(r_j)$, and $(p_k, p_l) \in C(r_i, r_j)$; also, $at(r_i, p_k)$ and $at(r_j, p_l)$ must hold at the beginning and during the task; we do not consider explicit effects, as they will be dealt with in a dedicated approach.

We associate with every task t a starting time point $\delta_s(t)$ and a duration $dur(t)$. We then define the end time of task t as $\delta_e(t) = \delta_s(t) + dur(t)$.

Constraints

Along with this formulation, we consider more elaborate constraints over the actions. In order to obtain realistic plans, we need to integrate some real-world knowledge based on models of the environment, and to be able to precisely synchronize communications between robots.

Visibility Constraints An external library provides us with the $O_i(z_j)$ and $C(r_i, r_j)$ static data. More precisely, visibilities for observation and communication are obtained from terrain models and ray-tracing algorithms, taking into account (optical) sensor ranges and occlusions.

Temporal Constraints on Motions One key point for tackling a realistic scenario is to model the duration of moves. We thus use an external function $dur(goto(r_i, p_j, p_k))$ which provides us which an estimation of the duration of motions. This function can be implemented using any path planning algorithms (LaValle 2006) based on precise terrain models. For our experiments, the external library we use implements an A^* algorithm on discrete terrain models taking into account robots' motion capabilities.

Temporal Constraints on Other Actions We consider a duration for observation and communication tasks. For now, these durations are constant, but this can be easily lifted up.

Constraints on Communications For an operational scenario, one important need for the operator is to regularly monitor the states of the robots and the progress of the mission. In our case, as communications are not always possible, we need to enforce periodical communications, in order to detect within a given time if a robot has a problem, for

instance if it is lost or blocked by an obstacle and unable to reach its objective.

In order to do so, we add a set of constraints on communications, using a centralized communication scheme:

- the operator has to be given a complete update periodically (every Δ minutes);
- communications with the operator are very limited;
- one of the robot is preferred for centralizing communications with other robots and the operator, because it has more motion capabilities than the others; for example an Autonomous Aerial Vehicle (AAV).

Dividing the Plan into Chunks

We solve the mission planning problem by dividing the plan into chunks. Each chunk corresponds to a sequence of observations followed by a communication between all teammates and the operator. Splitting plans into such chunks allows the operators to regularly receive a complete feedback concerning the state of each robot, which can make mission monitoring easier.

For making mission monitoring easier again, we build hierarchical plans, containing actions with different levels of abstractions. Operators may go deeper in the hierarchy of actions in order to understand what the robots are doing. We first define some abstract tasks.

Definition 1. *($goto\_and\_obs$) A $goto\_and\_obs$ abstract task is made of a movement task g and an observation task o:*

- $g = goto(r, p_i, p_j)$ *and* $o = obs(r, p_j, z_k)$*;*
- $r \in R$ *is the robot performing the tasks;*
- $p_j \in P(r)$ *is the point from where* $z_k \in Z$ *is observed;*
- $\delta_e(g) = \delta_s(o)$.

This abstract task is denoted as $goto\_and\_obs(r, p_i, p_j, z_k)$.

Definition 2. *($goto\_and\_com$) A $goto\_and\_com$ abstract task is made of two tasks g and c such that:*

- $g = goto(r_i, p_u, p_v)$ *and* $c = com(r_i, r_j, p_v, p_w)$*;*
- $r_i \in R$ *is the robot performing the tasks;*
- $p_v \in P(r_i)$ *is the point from where a communication is possible to* $p_w \in P(r_j)$*;*
- $\delta_e(g) = \delta_s(c)$.

This abstract task is denoted as $goto\_and\_com(r_i, r_j, p_u, p_v, p_w)$.

Now we can define a chunk as a sequence of $goto\_and\_obs$ for each robot, followed by synchronized $goto\_and\_com$ for all robots.

Definition 3 (Chunk)**.** *Let r_m denote the robot in charge of the centralization of the communications (m for "master"), and let op denote the operator. A chunk c is composed of:*

- *for each $r_i \in R$, a sequence of abstract tasks $goto\_and\_obs(r_i, p_j, p_{j+1}, z_k)$;*
- *for each $r_i \in R$, $i \neq m$, a task $goto\_and\_com(r_i, r_m, p_u, p_v, p_w)$ for robot r_i and a symmetric task $goto\_and\_com(r_m, r_i, p_x, p_w, p_v)$ for robot r_m; communications are synchronized, i.e. $\delta_s(com(r_i, r_m, p_v, p_w)) = \delta_s(com(r_m, r_i, p_w, p_v))$*
- *a task $goto\_and\_com(r_m, op, p_y, p_z, p_{op})$ at the end of the chunk, with p_{op} the point where the operator is.*

We define $dur(c)$ as the temporal distance between the beginning of the first $goto\_and\_obs$ task of the chunk and the end of the communication with the operator. The chunk is said to be valid if and only if $dur(c) \leq \Delta$, where Δ is the communication period set by the human operator.

The intuition is that a chunk is a refinement method in the HTN framework, augmented with temporal constraints. If communication period Δ is too small, then it will be impossible to include some observations into plans. A schematic graphical view of a chunk is given in Figure 1.

Definition 4 (Chunked Plan)**.** *A chunked plan $\mathcal{C}$ is a sequence of chunks $(c_i)_{i \geq 0}$ such that at the end of the execution we have $\forall z_k \in Z : toObs(z_k) = \bot$.*

The planning algorithm must ensure that given the chosen observations, the duration of any chunk does not exceed the required communication period (Δ). It must also minimize the number of chunks and, as a side effect, the total duration of the mission.

Supervising Execution

Plan Representation

Hierarchical and Temporal Tree of Tasks (HT^3) We represent the plan as a hierarchical tree of tasks along with their temporal execution windows. As said before, representing plans in such a way can make plans more readable for the operators. We follow the common hierarchical model of HTNs (Erol, Hendler, and Nau 1994) to model the plan, and we include some temporal information (Bresina et al. 2005) as well as allocation of tasks to robots.

Definition 5 (HT^3)**.** *An HT^3 plan $\mathcal{P} = (R, T, M, \mathcal{A}, \mathcal{I}, t_r)$ is defined by:*

- *a set of robots $R = \{r_i, 1 \leq i \leq n_r\}$;*
- *a set of tasks $T = T_e \cup_{\neq} T_a$, where T_e is a set of elementary tasks representing robots' actions, and T_a is a set of abstract tasks;*
- *a decomposition function (or set of methods) $M : T_a \rightarrow 2^{(T_a \cup T_e)} \times \{\rightarrow, \sim\}$ that associates with each abstract task a set of tasks st and a relation rel between the elements of st; rel is either a non-ordered relation ($\sim$, tasks of st can be executed in any order) or a sequence ($\rightarrow$, tasks of st must be executed in a specific order);*
- *an allocation function $\mathcal{A} : T \rightarrow 2^R$ that associates with each task a set of robots; for elementary tasks, only one robot is performing the task : $\forall t \in T_e, \#\mathcal{A}(t) = 1$;*
- *an interval function $\mathcal{I}$ that associates with each task t a time interval $\mathcal{I}(t) = [\delta^-, \delta^+]$ such that δ^- and δ^+ are the earliest and latest times to start the execution of the task;*
- *a root task $t_r \in T$.*

A plan $\mathcal{P}$ is valid only if it has no cycles (e.g., abstract tasks being child of themselves). It can then be represented as a tree, alternating tasks and methods. Note that contrarily to HTNs, HT^3 do not allow any choice in the decomposition of abstract tasks, which makes them directly executable.

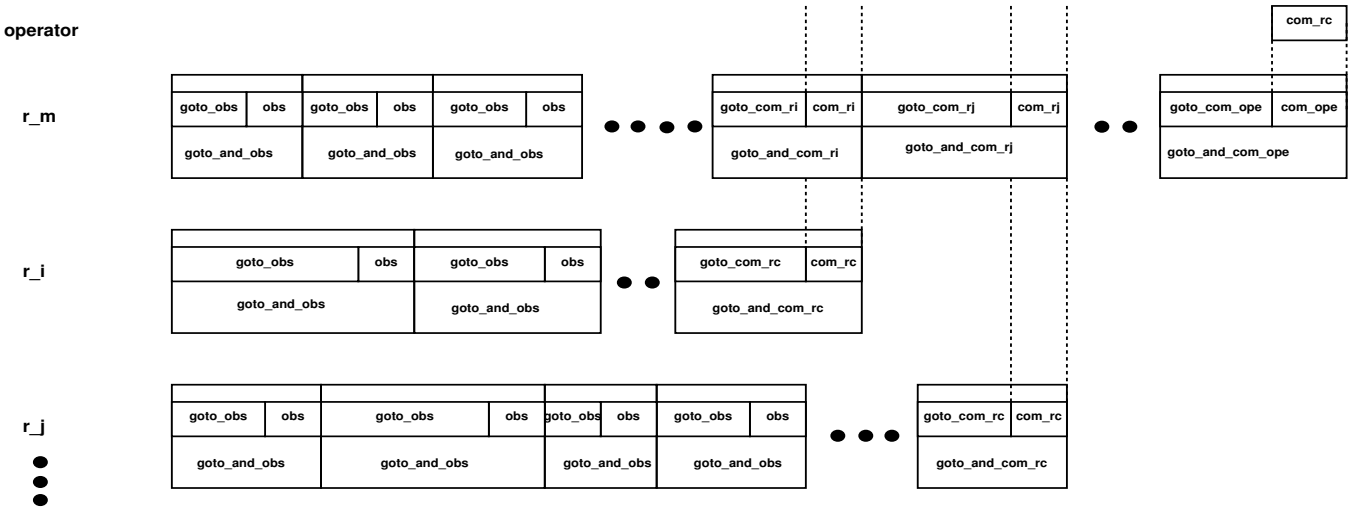


Figure 1: General scheme of a chunk

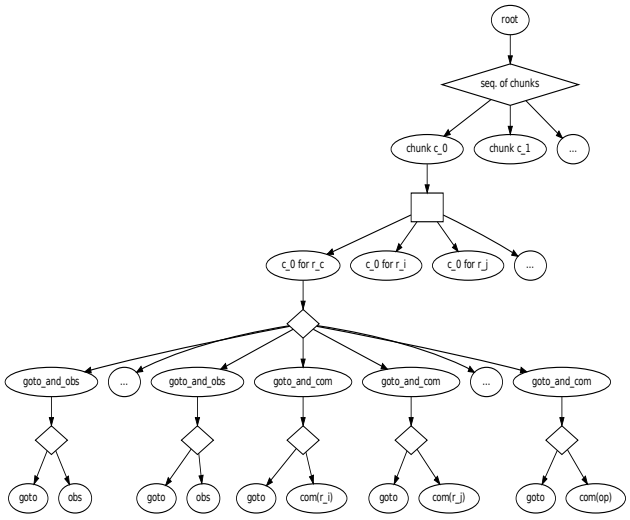


Figure 2: HT^3 built from a sequence of chunks. Ovals, diamonds, rectangles respectively represent tasks, sequential methods, unordered methods. For clarity, allocation and interval functions are not represented.

Chunked Plans as HT^3 Building an HT^3 plan (Def. 5) from a chunked plan (Def. 4) is quite straightforward: we first define a root task t_r, containing a sequential method with the sequence of chunks as children. Then each chunk is decomposed into a non-ordered method containing the tasks of each robot. Finally, the chunk for each robot is sequentially decomposed into abstract tasks $goto\_and\_obs$ and $goto\_and\_com$, themselves respectively decomposed into $goto$ and obs, and $goto$ and com elementary actions. Figure 2 shows the generic hierarchical pattern of a plan.

Algorithm 1 Execution of hierarchical and temporal plans

Require: $\mathcal{P}$ the global team plan, $r \in R$ the robot that locally executes the plan

1: **procedure** SCANTASK($t, r, \mathcal{P}$)
2: CHECKTIME($\mathcal{I}(t)$)
3: **if** $t \in T_e$ **then** EXECUTE(t)
4: **else** $\triangleright\ t \in T_a$
5: $st, rel \leftarrow M(t)$
6: Sort st according to rel
7: **for all** $t' \in st$ **do**
8: **if** $r \in \mathcal{A}(t')$ **then** SCANTASK($t', r, \mathcal{P}$)

Plan Execution

Each robot executes plan $\mathcal{P}$ following the procedure detailed in Algorithm 1. The plan is executed by each robot in a depth first (i.e. descending from abstract tasks to elementary tasks) and left first (i.e. enforcing the ordered relations between tasks) manner. Procedure SCANTASK is initially called with t_r (the root task) as a parameter.

Procedure CHECKTIME used at line 2 checks that the current execution time is in the allowed window of start times for task t. If the task is early, we wait for the earliest starting time of the task. If the task is on-time, we start it immediately. If the task is late, a repair is needed (see next paragraph).

Procedure EXECUTE used at line 3 triggers the execution of an elementary task on the robot. In practice, this is achieved by calling a service on the robot control architecture, in order to move the robot, make observations, or communicate with other robots.

Plan Repair

At execution, two kinds of disturbances are considered: lateness of tasks and execution failures of elementary actions.

Lateness During Execution In the CHECKTIME procedure (line 2), the task to execute may be detected as late, i.e. the latest time at which it should have started has passed. Then, the execution is inconsistent regarding the plan.

As communication is unreliable and not persistent, we cannot count on a global repair. Therefore, the repair process is driven by the key feature which ensures the success of the mission: the communication constraint between robots. We therefore try to secure communication tasks. For this purpose, the repair process is a simple hard-coded rule that removes $goto\_and\_obs$ tasks from the plan of the late robot until the current chunk becomes temporally consistent again. The new plan is executed until the end of the chunk, where a communication between all robots is possible (by using master robot r_m). At the end of the chunk, the operator knows the set of observations removed during repairs. He/she can then reallocate these observations and transmit the new plans to all robots through master robot r_m.

Failures of Execution Execution disturbances may also come from issues in the execution of tasks by the control architecture of the robot. For instance a mobile ground robot may face some obstacles during its movements. If it reaches the aimed point, the next task may be late due to the time needed to overcome the obstacle (previous lateness case). But the robot may even not reach the point, for instance because no path exists (at least in the robot map) to join this point. In these situations, which are considered as failures, we use some precomputed parametered local hierarchical plans that are adapted for repairing a set of predefined failures. When a failure occurs, depending on the task that has failed and of the cause of the failure, we replace the failing task with the appropriate local plan. More advanced plan repair techniques could be considered. In our experiments, three kinds of parametered local plans are considered:

- parametered backup trajectories to find communication opportunities with the master robot (to ask for help);
- map sharing between robots (to update the map of the lost robot);
- relative localization between robots (to update the position of the lost robot).

As these parametered local plans are represented as "local" HT[3], they can be directly inserted within the current plan and executed without any hack in the execution procedure.

Note that this paper does not provide repair rules for all possible lateness issues and all possible failures. It just describes a first step in the definition of hard-coded repair rules for the operational scenario we consider. Additional work on this point is left for future work. In particular, rules should be defined for tackling the case where master robot r_m itself fails, or cases in which removing observation tasks does not suffice to satisfy the chunk duration constraint again.

Planning

In this section, we describe the planning model implementation and the planning algorithm used to build mission plans offline.

Using a Generic Framework: InCELL

In order to state all temporal constraints of the problem and to handle them efficiently, we rely on the InCELL framework (Pralet and Verfaillie 2013). InCELL is inspired by *Constraint-based Local Search* (CLS (Hentenryck and Michel 2005)). In CLS, the user defines a model of its problem in terms of decision variables, constraints, and optimization criterion. For the multi-robot exploration mission, (1) *decision variables* correspond to the sequence of tasks to be performed by each robot, (2) *constraints* are either temporal constraints (activity durations and communication deadlines) or spatial constraints (zone observation and robot communication from possible locations), and (3) the *criterion* is to minimize the duration of the mission.

In CLS, the user also defines a local search procedure over complete variable assignments (where every decision variable is assigned). Such a local search procedure corresponds to a sequence of local moves. For the multi-robot exploration problem, examples of local moves are the addition/removal of a chunk and the addition/removal of an observation task for a robot inside a chunk.

Because the speed of local moves is one of the keys to local search success, CLS uses so-called *invariants*, which allow expressions and constraints to be quickly evaluated after each move. Invariants correspond to one-way constraints $[x_1, \ldots, x_n] \leftarrow f(y_1, \ldots, y_p)$, where $x_1, \ldots, x_n$ (the outputs) are variables whose assignment is a function of other variables $y_1, \ldots, y_p$ (the inputs). Invariant outputs are incrementally (quickly) reevaluated upon small changes in the inputs. In particular, InCELL models temporal constraints as invariants and uses incremental *Simple Temporal Network* (STN (Dechter, Meiri, and Pearl 1991)) techniques to efficiently maintain earliest/latest consistent times for activities.

Task Modeling InCELL allows tasks to be modeled based on the notion of *interval*. An interval is defined by two timepoints representing the start and the end of the task, and by one boolean variable representing the presence of the task in the plan. Elementary tasks of the multi-robot exploration problem (moves, observations, communications) are represented as InCELL intervals.

For the multi-robot exploration mission, goal activities are observations and communications, whereas setup activities are motions that occur between goal activities. Tasks goto_and_obs and goto_and_com represented in Figure 1 are modeled as intervals composed respectively of two subintervals [*goto_obs , obs*] and [*goto_com, com*]. Doing so, the hierarchical structure of plans is explicitly taken into account in the model.

The duration of each goal activity is constant in our case. The duration of a move towards a goal activity *Act* depends on the goal activity *Act* itself and on the goal activity *preAct* preceding the move. If the current and previous activities *Act* and *preAct* are provided, the InCELL invariant evaluator automatically calls the external terrain-aware function that computes estimations of move durations.

Chunk Modeling Beyond intervals, the multi-robot exploration model also uses the notion of sequence of contiguous intervals available in InCELL. Using this modeling

feature, it is easy to implement chunks as $n_r + 1$ sequences of contiguous intervals, one for each robot plus one for the operator. When inserting a new task in the middle of a sequence, one only needs to update the "previous activity" feature for the inserted task and for the task following it. The InCELL invariant in charge of managing temporal aspects then automatically updates earliest/latest times of all time-points of the problem, using incremental STN techniques. The chunk itself is a specialization of a temporal interval, so it also has start and end time-points.

Mission Constraints as InCELL Invariants The InCELL invariant managing temporal aspects takes as input all temporal constraints of the problem inside a chunk:

- equality of communication starts and communication ends for the two robots involved in a communication task;
- start time of the chunk equal to the start time of the first move of the chunk;
- end time of the communication with the operator equal to the end time of the chunk;
- bounded temporal distance between the start and the end of a chunk, in order to enforce communication period Δ.

Algorithm

On top of the InCELL model, we define an algorithm that allocates observation tasks within chunks. This algorithm combines: (1) a constructive greedy search phase, which produces a first exploration and communication plan; (2) a local search phase that improves on the plan found at the first phase.

Greedy Search We first use a greedy search that tries to put as many as possible observations inside a chunk, and when constraints are violated, a new chunk is created, and the process iterates until all zones are scheduled for observation. The pseudo-code is detailed in Algorithm 2.

In this algorithm, the main loop iterates until no more zones are to be observed. Inside this main loop, a chunk is first allocated at line 8, already containing $goto\_and\_com$ tasks, as described in the modeling section. Then a robot is selected at line 9. Several strategies are possible, we implemented a random choice giving more weight to slower robots, in order for them to be able to choose first their objectives, because in our scenarios all robots start from a close location, and we do not want the slower ones to go very far to achieve their first objectives. This procedure return $\emptyset$ when all robots are marked as full, that is when no robot can accept a new observation in the current chunk.

We then enter a second loop for filling current chunk c_i. First, the closest observation is selected for robot r_j at line 11. This is done on the basis of the navigation points $P(r_j)$ from which a zone that has not yet been observed can be observed, and of a heuristic implemented as an InCELL invariant (see below). We implemented both a simple Euclidean distance and the real distance as given by the external duration function used by the constraint checker.

The selected observation is inserted at the end of the chunk, just before the communication tasks (line 12); then,

Algorithm 2 Greedy search for allocating observation tasks

Require:
1: Z the set of zones to observe
2: R the set of robots
3: P the function giving navigation points
4: O the function giving points for observing zones
5: C the function giving pair of points for communication

6: **procedure** GREEDYSEARCH(Z,R,P,O,C)
7: **while** $Z \neq \emptyset$ **do**
8: c_i = ALLOCATENEWCHUNK
9: $r_j \leftarrow$ SELECTROBOT(R)
10: **while** $r_j \neq \emptyset$ **do**
11: $(z_k, p_l) \leftarrow$ SELECTOBS($Z, P(r_j), O, p_{l-1}$)
12: INSERT($c_i, (z_k, p_l), r_j$)
13: UPDATECOMLOC(c_i, C, P)
14: ok = EVALUATECONSTRAINTS
15: **if** $\neg$ ok **then**
16: CHANGECOMORDER(c_i, C, P)
17: UPDATECOMLOC(c_i, C, P)
18: ok = EVALUATECONSTRAINTS
19: **if** $\neg$ ok **then**
20: REVERTCHANGES
21: TAGASFULL(r_j, c_i)
22: **if** ok **then**
23: $Z \leftarrow Z \setminus \{z_k\}$
24: $r_j \leftarrow$ SELECTROBOT(R)

the communication locations are updated, by choosing for the slower robot the communication point that is the closest from the newly inserted observation task (line 13). A location for the other robot involved in the communication is chosen among the restricted possible ones, also based on its last objective so far. Then the InCELL model is evaluated at line 14.

If the insertion fails (lines 16-17), we try to swap some communication activities for master robot r_m (these activities are initially randomly ordered). We also try to change communication locations. If insertion is still impossible, we discard changes and mark the selected robot as full for the current chunk, meaning that it does not accept new observations in the current chunk.

If insertion is possible, we mark the zone as observed (line 23), we select a robot and we iterate the inner loop again. This loop ends as soon as all robots are full for the current chunk.

Heuristics as Invariants InCELL offers various high level invariants, including the *argmin* invariant used for selecting items into a set based on some criterion. We thus use a selection based on the argmin invariant to maintain the closest observation candidates for any robot, as well as the closest communication location candidates. These invariants allow a transparent heuristic computation, used in the SELECTOBS and UPDATECOMLOC procedures (lines 11 and 13).

Local Search While greedy search aims at minimizing the number of chunks (and thus the total number of mandatory communications), it has a major drawback: the last chunk is generally very inefficient, because of the priority given to the slower robots. It may involve only a few observations by the slower robot, while other faster robots do not have any. To overcome this issue, we perform a local search from the solution produced by the greedy search.

The lower levels local search operators are to *remove* an observation from any chunk (including the corresponding goto), and to *insert* a zone to observe anywhere in the plan, trying every observation location for a given zone.

Over these two basic local moves, we implemented:

- a *bestInsert* procedure, which tries in turn every possible insertion for a given zone to be observed (in every chunk), and returns an updated plan with the best insertion, in terms of earliest end time of the global mission;
- a *moveInSequence* procedure, that removes and tries to *bestInsert* every observation in turn, this for a given robot.

We also implemented higher level local moves, namely:

- *2-opt* (Croes 1958), which tries permutations of pairs of observations and inverts orders in-between;
- *relocate* (Salvelsbergh 1992), which first removes an observation from a robot, second tries to *bestInsert* it in the plan of another robot, third applies *2-opt*, and fourth applies *moveInSequence*; it does so for every observation, as long as there are improvements in the global earliest end time of the mission.

Other moves based on the computation of critical paths could also be used to compact the obtained schedules.

Evaluating Planning and Local Search

We first show some planning results on simple scenarios, to give a first idea of the efficiency of the approach.

First Example We first consider a 100 m $\times$ 100 m area containing 25 zones to be observed, and a temporal constraint on communications allowing the whole mission to be executed in only one chunk (namely 5 minutes). Figure 3a shows the trajectories for 1 Autonomous Aerial Vehicle (AAV) and 2 Autonomous Ground Vehicles (AGVs) after greedy search, without any local optimization. The earliest end time of the mission is 299.2 seconds. After local search, we non surprisingly obtain better trajectories, shown on Figure 3b, and the earliest end time of the mission lowers to 214.7 seconds. In the first case, the greedy search gives high priority to the two AGVs, so that they have respectively 11 and 12 objectives to observe, while the AAV, which is much faster, only has two. After local search, the AAV has 12 objectives, and the AGVs 7 and 6, respectively, leading to a much better distribution of tasks over time.

For this simple problem, on an Intel Core 2 Duo 3GHz-4GBRAM, the construction and initialization of the InCELL model took 780ms, and greedy search time is 161 ms. The local search took 2833 ms.

Other Examples A second example considers the very same problem but with a 3-minute constraint for chunk durations, so that the mission cannot be achieved in only one chunk (it takes 2). Before local search, the earliest end time is 358.6 seconds. After local search, it lowers to 291.2 seconds. In this case, the local search time raises to 9141 ms.

To give an idea for larger scenarios, if we consider 100 zones to observe, the greedy search takes 682 ms, while the local search raises to approximately two minutes. We thus have a first solution very quickly, and the local search can be used as an anytime algorithm, giving better solution as time passes. Figure 3c shows the evolution of the earliest end time of the mission wrt. computation times for 100 zones to observe.

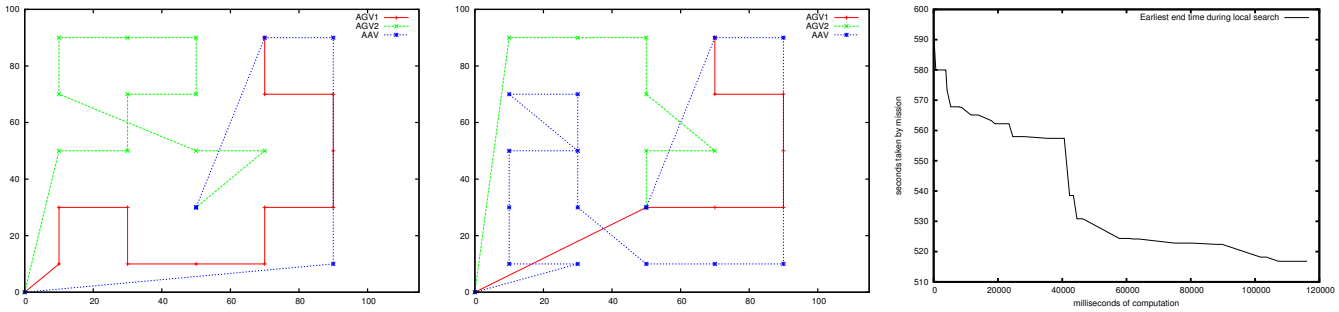
Complete Scenario

The complete scenario demonstrated fall 2013 involved one autonomous aerial vehicle and two autonomous ground vehicles. The area to explore was 28800 square meters wide, including 72 zones to observe. Figure 4a shows the area used for the experiments, for which a 3D model has been built in order to compute visibilities (for observations and communications) as well as durations of movements. The operator had to be given a complete update every 5 minutes (maximum size of a chunk). As flight authorizations are required to use our experimental platforms, only a few tests were performed in real conditions.

The InCELL model contains 31154 invariants. It is built and initialized in about 7 seconds. The resolution based on greedy and local search takes about 6 minutes, the best solution being obtained only after 3 minutes of computations. The huge difference with simple examples, that are about the same size, is mainly due to the calls to the precise computation of motion times, that are very slow. We implemented a cache in order to limit this impact.

Figure 4b shows the trajectories computed for this real-world mission. The duration of the mission plan obtained is 472 seconds. This duration is mainly due to the fact that the AAV has many more observations to do than the AGVs (the AGVs fill their objectives during the first chunk, while the AAV needs two). One can also notice that the observation tasks assigned to the two AGVs have a particular spatial repartition. This is due to the fact that their possible positions are very constrained: they must remain on the road since their navigation on the field can be problematic depending on the precise state of the terrain (height of the grass, wetness of the soil).

The execution procedure has been integrated on the three robots, as a separate program calling each robot's services to realize elementary actions. The aerial autonomous robot is a Yamaha Rmax helicopter that navigates based on a GPS sensor, and embeds a software architecture based on Orocos (Soetens and Bruyninckx 2005), in which actions are triggered by executing supervision state machines specified in the rFSM language (Klotzbücher and Bruyninckx 2012). The ground robots are Segway-based robots integrating lidar sensors for map building, cameras and inertial sensors used for localization, and embedding a Genom-based (Mallet,



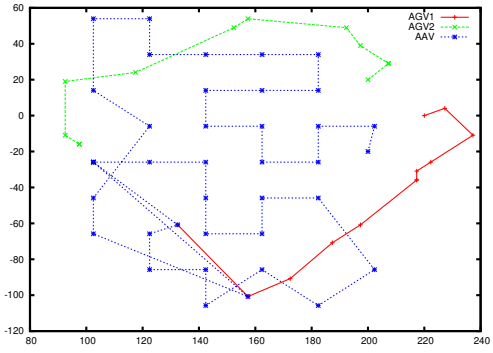
(a) Example 1 before local search (b) Example 1 after local search (c) Earliest end time (100 zones)

Figure 3: Plans produced on simple examples by the offline planning algorithm

Pasteur, and Herrb 2010) software architecture, in which actions are triggered by executing specific agents of the ROAR framework (Degroote and Lacroix 2011).

In fall 2013, we demonstrated the execution of the nominal plan, along with the management of some failures (one of the two ground robots get lost and, following hard-coded repair rules, asked for the Rmax helicopter to map the environment around the ground robot). We also demonstrated the replanning process including reallocation of the unexplored zones. We plan to do some complementary experiments in the future, in order to demonstrate the management of lateness of execution, by introducing disturbances at any moment in the mission execution.

(a) Map of the experiments area



(b) Trajectories after greedy and local search

Figure 4: Real-world scenario

Related Work

Some works focus on allocating exploration tasks to several robots, but either do not consider communication constraints (using frontier-like exploration (Hourani, Hauck, and Jeschke 2013) or a segmentation of the environment (Wurm, Stachniss, and Burgard 2008)), or try to maintain communication capabilities at any time by deploying a network infrastructure (Pei and Mutka 2012; Abichandani, Benson, and Kam 2013). Some approaches use opportunistic communications to optimize the plan (Sung, Ayanian, and Rus 2013), but do not enforce them. These approaches do not consider time constraints between tasks, and when synchronization is explicitly modeled, it is focused on spatial synchronization (Coltin and Veloso 2012). Other approaches propose mechanisms to maintain the plan consistency at execution. In (Kaminka et al. 2007), robots regularly communicate to update temporal constraints between their tasks in order to maintain the global plan consistency. In the exploration mission considered in (Korsah et al. 2012), offline task scheduling and online plan flexibility are combined, and robots adapt their plans by exchanging messages for satisfying task constraints again. However, associated communication tasks are not explicitly included within plans. Moreover, these tasks are considered as not subject to failures.

In another direction, multi-robot task scheduling deals with time constraints such as task precedence or synchronization (Zhang and Parker 2013). In (Ponda et al. 2010; Luo, Chakraborty, and Sycara 2013), tasks are scheduled using an auction algorithm to minimize their delays. Mixed Integer Linear Programming (MILP) is used in (Koes, Nourbakhsh, and Sycara 2006) to solve task allocation problems with constraints modeled using Allen's algebra, and in (Mathew, Smith, and Waslander 2013) to find trajectories of robots that must meet the already planned trajectories of robots to be recharged. In these works, once tasks are scheduled, no communication occurs to share information and maintain plan consistency.

In probabilistic domains, (Wu, Zilberstein, and Chen 2011) proposes to broadcast history of actions and observations when an inconsistency is detected between the current observation and the belief state of a local POMDP. (Matignon, Jeanpierre, and Mouaddib 2012) uses

a DecMDP model with communication uncertainty, while (Spaan and Melo 2008) defines local interactions within a so-called IDMG model. While these approaches generate policies which include communication tasks, they do not integrate temporal constraints between tasks.

Conclusion and Future Work

We presented in this paper the high-level aspects of a complete multi-robot exploration mission, from mission modeling to execution and repair, including planning algorithms taking into account various constraints. We rely on the robust generic framework InCELL for planning algorithms, as well as a HTN-like paradigm (augmented with temporal aspects) for solution scheme, distributed execution and repair.

This work will continue in several directions. First, at model level, we want to take into account resources like energy, but also to do time-dependent scheduling. A typical case is that for optical sensors, it might be useful to be able to select observations locations with respect to the precise time of the action, in order not to have sun on sight for instance. Tasks should also be possibly done in parallel. For instance an observation should not be simply to take a picture of a location from a given position, but grab a whole video flow for a given time, this while moving. Similarly, communication and observation could be performed in parallel. We also would like to use a good criterion for plans, including much more information than only the earliest global end time. This includes a trade-off with the flexibility, but also variable costs of motions, variable values of interest for zones. Finally, we would like to stop discretizing the mission over predefined observation zones.

At planning level, we are also working on a very different approach for planning with temporal HTNs, that should be more generic and allow a user (maybe directly the operator) to specify the chunk-like decomposition, instead of the current fixed one.

At execution and especially at repair level, we plan to make the planning model "alive" on the different robots and feed it with real execution times in order to detect temporal inconsistencies much earlier, and replan as soon as possible. While deferring observation tasks for a given robot is an easy solution, ensuring global consistency and quality of the plan is quite a challenge. As previously mentioned, we plan to extend the set of failure cases which can be handled by the repair process. Last, we would also like to integrate the hand-written repair methods into the main planning loop, instead of leaving them only at the supervision level.

References

Abichandani, P.; Benson, H.; and Kam, M. 2013. Robust Communication Connectivity for Multi-Robot Path Coordination using Mixed Integer Nonlinear Programming: Formulation and Feasibility Analysis. In *International Conference on Robotics and Automation (ICRA)*.

Bresina, J. L.; Jónsson, A. K.; Morris, P. H.; and Rajan, K. 2005. Activity planning for the mars exploration rovers. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 40–49.

Coltin, B., and Veloso, M. 2012. Optimizing for Transfers in a Multi-Vehicle Collection and Delivery Problem. In *International Symposium on Distributed Autonomous Robotic Systems (DARS)*.

Croes, G. A. 1958. A method for solving traveling salesman problems. *Operations Research* 6:791–812.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal Constraint Networks. *Artificial Intelligence* 49:61–95.

Degroote, A., and Lacroix, S. 2011. ROAR: Resource oriented agent architecture for the autonomy of robots. In *International Conference on Robotics and Automation (ICRA)*.

Erol, K.; Hendler, J.; and Nau, D. 1994. HTN Planning: Complexity and Expressivity. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Gateau, T.; Lesire, C.; and Barbier, M. 2013. HiDDeN: Cooperative Plan Execution and Repair for Heterogeneous Robots in Dynamic Environments. In *International Conference on Intelligent Robots and Systems (IROS)*.

Hentenryck, P. V., and Michel, L. 2005. *Constraint-based Local Search*. MIT Press.

Hourani, H.; Hauck, E.; and Jeschke, S. 2013. Serendipity Rendezvous as a Mitigation of Explorations Interruptibility for a Team of Robots. In *International Conference on Robotics and Automation (ICRA)*.

Kaminka, G.; Yakir, A.; Erusalimchik, D.; and Cohen-Nov, N. 2007. Towards collaborative task and team maintenance. In *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*.

Klotzbücher, M., and Bruyninckx, H. 2012. Coordinating Robotic Tasks and Systems with rFSM Statecharts. *Journal of Software Engineering for Robotics (JOSER)* 3(1):28–56.

Koes, M.; Nourbakhsh, I.; and Sycara, K. 2006. Constraint optimization coordination architecture for search and rescue robotics. In *International Conference on Robotics and Automation (ICRA)*.

Korsah, A.; Kannan, B.; Browning, B.; Stentz, A.; and Dias, B. 2012. xBots: An approach to generating and executing optimal multi-robot plans with cross-schedule dependencies. In *International Conference on Robotics and Automation (ICRA)*.

LaValle, S. 2006. *Planning Algorithms*. Cambridge University Press.

Luo, L.; Chakraborty, N.; and Sycara, K. 2013. Distributed Algorithm Design for Multi-Robot Task Assignment with Deadlines for Tasks. In *International Conference on Robotics and Automation (ICRA)*.

Mallet, A.; Pasteur, C.; and Herrb, M. 2010. GenoM3: Building middleware-independent robotic components. In *International Conference on Robotics and Automation (ICRA)*.

Mathew, N.; Smith, S. L.; and Waslander, S. L. 2013. A Graph-Based Approach to Multi-Robot Rendezvous for Recharging in Persistent Tasks. In *International Conference on Robotics and Automation (ICRA)*.

Matignon, L.; Jeanpierre, L.; and Mouaddib, A.-I. 2012. Coordinated Multi-Robot Exploration Under Communication Constraints Using Decentralized Markov Decision Processes. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Pei, Y., and Mutka, M. 2012. Steiner traveler: Relay deployment for remote sensing in heterogeneous multi-robot exploration. In *International Conference on Robotics and Automation (ICRA)*.

Ponda, S.; Redding, J.; Choi, H.-L.; How, J.; Vavrina, M.; and Vian, J. 2010. Decentralized Planning for Complex Missions with Dynamic Communication Constraints. In *American Control Conference (ACC)*.

Pralet, C., and Verfaillie, G. 2013. Dynamic online planning and scheduling using a static invariant-based evaluation model. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Salvelsbergh, M. W. P. 1992. The vehicle routing problem with time windows: Minimizing route duration. *Journal on Computing* 4:146–154.

Soetens, P., and Bruyninckx, H. 2005. Realtime hybrid task-based control for robots and machine tools. In *International Conference on Robotics and Automation (ICRA)*.

Spaan, M., and Melo, F. 2008. Interaction-driven Markov games for decentralized multiagent planning under uncertainty. In *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*.

Sung, C.; Ayanian, N.; and Rus, D. 2013. Improving the Performance of Multi-Robot Systems by Task Switching. In *International Conference on Robotics and Automation (ICRA)*.

Wu, F.; Zilberstein, S.; and Chen, X. 2011. Online planning for multi-agent systems with bounded communication. *Artificial Intelligence* 175:487–511.

Wurm, K.; Stachniss, C.; and Burgard, W. 2008. Coordinated multi-robot exploration using a segmentation of the environment. In *International Conference on Intelligent Robots and Systems (IROS)*.

Zhang, Y., and Parker, L. 2013. Multi-Robot Task Scheduling. In *International Conference on Robotics and Automation (ICRA)*.