

Configuration Planning with Multiple Dynamic Goals*

M. Di Rocco and F. Pecora and P.K. Sivakumar and A. Saffiotti

Center for Applied Autonomous Sensor Systems, Örebro University, SE-70182 Sweden
{modo, fpa, pkr, asaffio}@aass.oru.se

Abstract

We propose an approach to configuration planning for robotic systems in which plans are represented as constraint networks and planning is defined as search in the space of such networks. The approach supports reasoning about time, resources, and information dependencies between actions. In addition, the system can leverage the flexibility of such networks at execution time to support dynamic goal posting and re-planning.

1 Introduction

Planning in AI was born with robots (Fikes and Nilsson 1972). Since then, the field of AI planning has progressed enormously. Yet, if you look inside a typical autonomous robot today you will see little evidence of this progress, and you may suspect that planning in AI has focused on issues which are not the main concerns of robot builders. We share this suspicion. If you ever tried to program a robot to accomplish tasks in unstructured, everyday environments, what you expected from an AI planner was probably the ability to: (i) reason about the physical aspects of the domain, like time, space, information requirements and resources; (ii) dynamically accommodate new goals, which might interact with the current ones; (iii) generate plans that enable some degree of flexibility during execution; and (iv) deal with multiple robots and devices, and with their physical and logical dependencies. While planners exist that exhibit some of the above features (Ghallab and Laruelle 1994; Parker 1998; Knight et al. 2001; Doherty, Kvarnström, and Heintz 2009; Barreiro et al. 2012) you will have a hard time to find a single system that gives you all of them. This paper is a first step toward the construction of such a system.

We present a *configuration planner*: a system that generates *configuration plans* for robotic systems that consist of mixed ecologies of robots and devices (Saffiotti et al. 2008). Such plans specify what actions the different actors should execute and what information they should exchange in order to perform a given task. Configuration planners have been proposed before (Parker and Tang 2006;

Lundh, Karlsson, and Saffiotti 2008), but they cannot deal with multiple dynamic goals, time or resources. By contrast, our planner can accommodate time, resources, multiple dynamic goals, and flexible execution. This is achieved by: (1) representing a (configuration) plan as a constraint network; (2) defining the configuration planning process as search in the space of such networks; and (3) sharing the constraint network between the planner and the executor. The first two steps allow for the integration of multiple facets in the planning problem, e.g., time, resources, and information dependencies; the third one allows for flexible execution and dynamic goal posting.

2 Representation

Our approach is grounded on the notion of *state variable*, which models elements of the domain whose state in time is represented by a symbol. State variables, whose domains are discrete sets, represent parts of the real world that are relevant for the configuration planner's decision processes. These include the actuation and sensing capabilities of the robotic systems, and the various aspects of the environment that are meaningful. For instance, a state variable can represent the capabilities of a physical device such as a robot, whose meaningful states might be “navigating”, “grasping” and “idle”. Similarly, a state variable can represent the interesting states of the environment, e.g., the state of a light which can be “on”, “off” or “broken”. Let S be the set of state variables in a given application scenario.

Some devices require resources when they are in given states. We employ the concept of *reusable resource*, i.e., a resource with a limited capacity which is fully available when not required by a device. An example of reusable resource is power: a maximum wattage is available, and devices can simultaneously require power so long as the sum of requirements is less than the maximum power. We denote with \mathcal{R} the set of all resource identifiers. Given a resource $R \in \mathcal{R}$, its capacity is a value $\text{Cap}(R) \in \mathbb{N}$.

Finally, devices in our domain may serve the purpose of providing or requiring certain *information contents*. For instance, a software component may require range data from a laser range finder, and provide localization information. Let the set of all information contents be denoted IC .

*This work was funded by the EC Seventh Framework Programme (FP7/2007-2013) grant agreement no. 288899 Robot-Era. Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

2.1 Representing Configuration Plans and Goals

We employ *activities* to represent predicates on the possible evolution of state variables:

Definition 1 An activity a is a tuple $(x, \mathbf{v}, I, u, \text{In}, \text{Out})$, where

- $x \in \mathcal{S}$ is a state variable;
- \mathbf{v} is a possible state of the state variable x ;
- $I = [I_s, I_e]$ is a flexible temporal interval within which the activity can occur, where $I_s = [l_s, u_s]$, $I_e = [l_e, u_e]$, $l_{s/e}, u_{s/e} \in \mathbb{N}$ represent, respectively, an interval of admissibility of the start and end times of the activity;
- $u : \mathcal{R} \rightarrow \mathbb{N}$ specifies the resources used by the activity;
- $\text{In} \subseteq \text{IC}$ is a set of required information contents;
- $\text{Out} \subseteq \text{IC}$ is a set of provided information contents.

Henceforth, we indicate with $(\cdot)^{(a)}$ an element of the five-tuple pertaining to activity a . The pair $(x^{(a)}, \mathbf{v}^{(a)})$ of an activity a asserts a particular state \mathbf{v} of the state variable x ; the flexible temporal interval $I^{(a)}$ represents possible temporal intervals of occurrence of the state $\mathbf{v}^{(a)}$ of state variable $x^{(a)}$. Note also that a pair of activities (a, b) is *possibly concurrent* if $I^{(a)} \cap I^{(b)} \neq \emptyset$. A pair (a, b) of possibly concurrent activities thus indicates that state variables $x^{(a)}$ and $x^{(b)}$ can be, respectively, in states $\mathbf{v}^{(a)}$ and $\mathbf{v}^{(b)}$ at the same time.

Unspecified parameters of an activity are indicated with (\cdot) — e.g., $(x, \cdot, I, u, \text{In}, \text{Out})$ indicates a predicate asserting that state variable x can be in any state during interval I , using resources as indicated by u , etc.

Activities can be bound by *temporal constraints*, which restrict the occurrence in time of the predicates. Temporal constraints can be of two types:

- *Binary temporal constraints* in the form $a C b$ prescribe the relative placement in time of activities a, b — these constraints are relations in Allen’s Interval Algebra (Allen 1984), and restrict the possible bounds for the activities’ flexible temporal intervals $I^{(a)}$ and $I^{(b)}$;
- *Unary temporal constraints* in the form $C a$ prescribe bounds on the start or end time of an activity a — these constraints are commonly referred to as *release time constraints* and *deadlines*.

Allen’s interval relations are the thirteen possible temporal relations between intervals, namely “precedes” (p), “meets” (m), “overlaps” (o), “during” (d), “starts” (s), “finishes” (f), their inverses (e.g., p^{-1}), and “equals” (\equiv).

When state variables are used to represent a system, the overall temporal evolution of such system is described by a *constraint network*:

Definition 2 A constraint network is a pair $(\mathcal{A}, \mathcal{C})$, where \mathcal{A} is a set of activities and \mathcal{C} is a set of constraints among activities in \mathcal{A} .

A constraint network can be used to represent a *configuration plan*. Configuration plans are said to be *feasible* if they are consistent with respect to the resource, state, and temporal requirements. Specifically,

Definition 3 A configuration plan $(\mathcal{A}, \mathcal{C})$ is feasible iff:

- the constraint network is temporally consistent, i.e., there exists at least one allocation of fixed bounds to intervals such that all temporal constraints are satisfied;
- activities do not over-consume resources, i.e., $\sum_{a \in A} u^{(a)}(R) \leq \text{Cap}(R)$, $\forall R \in \mathcal{R}$, where $A \subseteq \mathcal{A}$ is a set of possibly concurrent activities;
- activities do not prescribe that state variables assume different states in overlapping temporal intervals, i.e., $\mathbf{v}^{(a)} \neq \mathbf{v}^{(b)}$, $\forall (a, b) \in A \times A : x^{(a)} = x^{(b)}$, where $A \subseteq \mathcal{A}$ is a set of possibly concurrent activities.

A goal for a configuration planning problem is also represented as a constraint network, therefore expressing temporal, resource, state and information requirements. Typically, a goal (A_g, C_g) is an under-specified configuration plan. Initial conditions are feasible sub-networks of a goal. Maintaining constraints on the configuration plan rather than committing to a specific configuration plan directly enables dynamic goal posting, execution monitoring, and incremental adaptation to contingent events, as we show in Section 3.

2.2 Domain

Given a goal (A_g, C_g) and a configuration plan $(\mathcal{A}, \mathcal{C})$ which contains the goal, the feasibility of the configuration plan is not a sufficient condition for achieving the goal. This is because feasibility does not enforce information and causal requirements. The way these requirements are to be enforced depends on a *domain*:

Definition 4 A configuration planning problem is a pair $((A_g, C_g), \mathcal{D})$, where (A_g, C_g) is a goal constraint network, and \mathcal{D} is a domain. The domain is a collection of operators, which describe the information and causal dependencies between activities.

Definition 5 An operator is a pair $(a, (A, C))$ where

- $a = (x, \mathbf{v}, \cdot, \cdot, \cdot, \text{Out})$ is the head of the operator;
- $A = A_p \cup A_e \cup \{a\}$ is a set of activities, where
 - A_p is a set of preconditions, i.e., requirements, in terms of state variable values, information input, and resource usage, needed to achieve the state $\mathbf{v}^{(a)}$ of state variable $x^{(a)}$ and to produce $\text{Out}^{(a)}$;
 - A_e is a set of effects, i.e., state variable values entailed by the achievement of state $\mathbf{v}^{(a)}$ of state variable $x^{(a)}$;
- C is a set of temporal constraints among activities in A .

Computing a configuration plan consists in selecting and instantiating operators from the domain into the goal constraint network. Unlike in classical planning (Ghallab, Nau, and Traverso 2004), the relevance of an operator (γ^{-1}) is not determined by unifying effects with sub-goals, rather by the unification of an operator’s head with a sub-goal. The head of an operator is a non-ground activity which describes the value of a state variable and the information provided as a result of applying the operator. Preconditions and effects are nevertheless modeled, as their presence in the constraint network is dealt with differently at execution time (see Section 4).

An operator can be used to specify the information requirements needed for achieving a particular functionality. For instance, the `MoveFromTo` operator, which does not provide any information content, requires the current position of the robot:

$$\begin{aligned} a &= (\text{MoveFromTo}, \text{kitchen_livingroom}, \cdot, \cdot, \cdot, \emptyset) \\ A_p &= \{a_1, a_2\}, A_e = \{a_3\}, \text{ where} \\ a_1 &= (\cdot, \cdot, \cdot, \cdot, \cdot, \{\text{position}\}) \\ a_2 &= (\text{RobotLocation}, \text{kitchen}, \cdot, \cdot, \cdot, \cdot) \\ a_3 &= (\text{RobotLocation}, \text{livingroom}, \cdot, \cdot, \cdot, \cdot) \\ C &= \{a \text{ d } a_1, a \text{ m}^{-1} a_2, a \text{ m } a_3\} \end{aligned}$$

The head of the operator is a predicate on the functionality `MoveFromTo`. The operator is considered relevant when the constraint network contains an activity $(\text{MoveFromTo}, \text{kitchen_livingroom}, \cdot, \cdot, \cdot, \cdot)$, i.e., when a (sub-)goal stating that the robot must move from the kitchen to the living room is present in the network. The operator also prescribes the temporal relations that must exist between the activities, namely that the `MoveFromTo` functionality should occur during the availability of the position data ($a \text{ d } a_1$), that it should be met by the precondition of the robot being in the kitchen ($a \text{ m}^{-1} a_2$), and that it meets the effect of the robot being in the living room ($a \text{ m } a_3$).

An operator can also be used to represent a means to achieve certain information requirements. For example, the operator

$$\begin{aligned} a &= (\text{VisualSLAM}, \text{running}, \cdot, u(\text{CPU}) = 10, \cdot, \{\text{position}\}) \\ A_p &= \{a_1, a_2\}, A_e = \emptyset, \text{ where} \\ a_1 &= (\cdot, \cdot, \cdot, \cdot, \cdot, \{\text{range_data}\}) \\ a_2 &= (\cdot, \cdot, \cdot, \cdot, \cdot, \{\text{ref_frame}\}) \\ C &= \{a \text{ d } a_1, a \text{ m}^{-1} a_2\} \end{aligned}$$

specifies one way to achieve the necessary information requirement (position) for the `MoveFromTo` operator, namely through visual SLAM. This localization functionality requires (1) a functionality which provides range data, (2) a reference frame for the computation of the position estimate, and (3) 10% of the CPU resource. Also, the operator states that range data should be available during the entire duration of the localization process, and that the reference frame is needed at the beginning of the process.

The above operator does not specify how to obtain the needed information inputs. For instance, the range data might be provided through the following operator:

$$\begin{aligned} a &= (\text{StereoCamDriver}, \text{on}, \cdot, u(\text{Cam1}) = 1, \cdot, \{\text{range_data}\}) \\ A_p &= \{a_1\}, A_e = \emptyset, \text{ where } a_1 = (\text{Light}, \text{on}, \cdot, \cdot, \cdot, \cdot) \\ C &= \{a \text{ d } a_1\} \end{aligned}$$

An operator may also specify that the reference frame is obtainable by invoking a functionality of the stereo camera's pan-tilt unit:

$$\begin{aligned} a &= (\text{PanTilt}, \text{return_ref_frame}, \cdot, \cdot, \cdot, \{\text{ref_frame}\}) \\ A_p &= \emptyset, A_e = \emptyset, C = \emptyset \end{aligned}$$

The above operators can be applied to obtain a configura-

tion plan from the following goal constraint network:

$$\begin{aligned} A &= \{a_0 = (\text{MoveFromTo}, \text{kitchen_livingroom}, I_0, \cdot, \cdot, \cdot)\}, \\ C &= \emptyset \end{aligned}$$

Specifically, a particular application of the above operators may refine the given constraint network to the following:

$$\begin{aligned} A &= \{a_0 = (\text{MoveFromTo}, \text{kitchen_livingroom}, I_0, \emptyset, \emptyset, \emptyset) \\ a_1 &= (\text{VisualSLAM}, \text{running}, I_1, u(\text{CPU}) = 10, \\ &\quad \{\text{ref_frame}, \text{range_data}\}, \{\text{position}\}) \\ a_2 &= (\text{RobotLocation}, \text{kitchen}, I_2, \emptyset, \emptyset, \emptyset) \\ a_3 &= (\text{RobotLocation}, \text{livingroom}, I_3, \emptyset, \emptyset, \emptyset) \\ a_4 &= (\text{StereoCamDriver}, \text{on}, I_4, \\ &\quad u(\text{Cam1}) = 1, \emptyset, \{\text{range_data}\}) \\ a_5 &= (\text{PanTilt}, \text{return_ref_frame}, I_5, \emptyset, \\ &\quad \emptyset, \{\text{ref_frame}\}) \\ a_6 &= (\text{Light}, \text{on}, I_6, \emptyset, \emptyset, \emptyset)\}, \\ C &= \{a_0 \text{ d } a_1, a_0 \text{ m}^{-1} a_2, a_0 \text{ m } a_3, a_1 \text{ d } a_4, a_1 \text{ m } a_5, a_4 \text{ d } a_6\} \end{aligned}$$

This network represents a temporally consistent configuration plan in which resources are never used beyond their capacity, and state variables are never required to assume different values in overlapping temporal intervals. The plan is therefore feasible. Furthermore, the plan contains activities providing the required information contents as determined by the operators in the domain. However, not all causal dependencies are necessarily achieved by construction. If, e.g., the initial condition does not state that the light is on, the configuration planner would regard the activity a_6 as yet another sub-goal to satisfy, and might do so by applying the following operator:

$$\begin{aligned} a &= (\text{Light}, \text{on}, \cdot, \cdot, \cdot, \cdot) \\ A_p &= \emptyset, A_e = \{a_1\}, \text{ where } a_1 = (\text{LightController}, \text{on}, \cdot, \emptyset, \cdot, \cdot) \\ C &= \{a \text{ p}^{-1} a_1\} \end{aligned}$$

This operator models an actuation process (`Light` represents an environment variable), and its application would refine the configuration plan by adding an activity $a_7 = (\text{LightController}, \text{on}, I_7, \emptyset, \emptyset, \emptyset)$ to the network, along with the constraint $a_6 \text{ p}^{-1} a_7$, prescribing that the `LightController` be in state `on` before the light is required to be on. Note that the light control functionality has no information requirements ($\text{In}^{(a_1)} = \emptyset$).

3 Constraint-Based Search

The planning process used in our approach is incremental in nature, and yields a refined constraint network, which itself represents a feasible configuration plan which achieves the given goal. The resulting constraint network represents one or more temporal evolutions of the state variables that guarantee the achievement of the goal under nominal conditions. Feasible and goal-achieving configuration plans are obtained in our approach by means of four interacting solvers: **Temporal solver.** The temporal consistency of the constraint network is checked through temporal constraint propagation by means of a Simple Temporal Problem (STP) (Dechter,

Meiri, and Pearl 1991) solver. The solver propagates temporal constraints to refine the bounds $[l_s, u_s], [l_e, u_e]$ of the activities in the network, and returns failure if and only if temporally consistent bounds cannot be found.

Resource scheduler. This solver enforces that resources are never over-consumed. The maximum capacities of resources restrict which activities can occur concurrently, and this solver posts temporal constraints to the constraint network enforcing that over-consuming peaks of activities are avoided (Cesta, Oddi, and Smith 2002).

State variable scheduler. State variable scheduling ensures that activities do not prescribe conflicting states in overlapping intervals. Similarly to the resource scheduler, this solver posts temporal constraints which impose a temporal separation between conflicting activities.

Information dependency reasoner. Operators model the information dependencies between functionalities¹. This solver instantiates into the constraint network relevant operators (in the form of activities and temporal constraints) so as to enforce the information dependencies.

Causal planner. Operators in the domain also model causal dependencies between states. This solver instantiates into the constraint network relevant operators (in the form of activities and temporal constraints) so as to enforce the causal dependencies of the configuration plan.

As noted, resource over-consumption and multiple concurrent states are averted by imposing temporal constraints which sequence potentially concurrent activities; trivially, there are alternative sequencing decisions that can be made to resolve such a conflict, e.g., enforcing $a \text{ p } b$ or $a \text{ p}^{-1} b$. Also operator selection is subject to alternative choices, as more than one operator may provide the necessary information output and/or support the necessary causal dependency (e.g., the presence of light in the environment may be obtained as a result of invoking the light controller or by opening the blinds.) Note that only temporal feasibility enforcement is not subject to multiple choices, as the problem is tractable. In our approach, all requirements which may entail alternative courses of action are seen as *decision variables* in a high-level Constraint Satisfaction Problem. Given a decision variable d , its possible values constitute a finite domain $\delta^d = \{(A_r^d, C_r^d)_1, \dots, (A_r^d, C_r^d)_n\}$, whose values are alternative constraint networks, called *resolving constraint networks*. The individual solvers are used to determine resolving constraint networks $(A_r^d, C_r^d)_i$, which are iteratively added to the goal constraint network (A_g, C_g) .

In order to search for resolving constraint networks, we employ a systematic search (see Algorithm Backtrack), which occurs through standard CSP-style backtracking. The decision variables are chosen according to a variable ordering heuristic h_{var} (line 1); the alternative resolving constraint networks are chosen according to a value ordering heuristic h_{val} (line 5). The former decides which (sub-)goals to attempt to satisfy first, e.g., to support a functionality by applying another operator, or to resolve a scheduling conflict. The latter decides which value to attempt first, e.g., whether

¹In our approach, the domain is such that information dependencies constitute an acyclic propositional Horn theory.

Function Backtrack (A_g, C_g) : success or failure

```

1  $d \leftarrow \text{Choose}((A_g, C_g), h_{\text{var}})$ 
2 if  $d \neq \emptyset$  then
3    $\delta^d = \{(A_r^d, C_r^d)_1, \dots, (A_r^d, C_r^d)_n\}$ 
4   while  $\delta^d \neq \emptyset$  do
5      $(A_r^d, C_r^d)_i \leftarrow \text{Choose}(d, h_{\text{val}})$ 
6     if  $(A_g \cup A_r^d, C_g \cup C_r^d)$  is temporally consistent then
7       return Backtrack  $(A_g \cup A_r^d, C_g \cup C_r^d)$ 
8      $\delta^d \leftarrow \delta^d \setminus \{(A_r^d, C_r^d)_i\}$ 
9   return failure
10 return success

```

to prefer one operator over another. Note that adding resolving constraint networks may entail the presence of new decision variables to be considered.

The possible values for resource contention or unique state decision variables are temporal precedences among activities. Values for information decision variables are ground operators, as shown in the previous Section. Lastly, values for causal decision variables are either ground operators, or *unifications* with activities that already exist in the constraint network. Two activities a and b can be unified if $x^{(a)} = x^{(b)} \wedge v^{(a)} = v^{(b)}$. Unifications are enforced by imposing a temporal equality constraint $a \equiv b$ among the activities. Supporting unification is obviously necessary to allow the search to build on previously added activities — e.g., leveraging that the light has already been turned on to support a previously branched-upon causal dependency. More importantly, unification also allows to accommodate on-going sensing and execution monitoring processes during configuration planning. For instance, activity $a = (\text{Light}, \text{on}, I^{(a)}, \emptyset, \emptyset, \emptyset)$ could be supported by unification with an activity $a_{\text{sensed}} = (\text{Light}, \text{on}, [[0, 0][13, 13]], \emptyset, \emptyset, \emptyset)$ which models the temporal interval within which a light source was sensed by a sensor in the environment.

4 Plan Execution and Dynamic Plan Update

The ability to support on-line sensing is directly enabled by the constraint-based representation: sensing is reduced to dynamically updating the constraint network with new activities and constraints representing the sensed state of the environment; the same mechanism also supports prediction (i.e., “sensing in the future”) and to other on-line plan modifications, such as temporal delays and dynamically posted goal constraint networks.

Our approach is based on the alternation of planning and plan execution monitoring. The former consists of the planning procedure shown above. The latter consists of two processes, *sensing* and *plan update*. The sensing process adds to the constraint network activities and temporal constraints representing the current view of the environment as provided by sensors. The plan update process maintains and updates temporal constraints which bound on-going activities (sensed states or functionalities in execution) with the current time. This is done in $O(n^2)$. Also, this process imposes constraints that verify the existence of preconditions and trigger the manifestation of effects contained in the plan. Specifically, the presence of a precondition is verified by at-

tempting to unify the activity representing the precondition with a sensed activity. If the unification is not possible, the precondition is delayed by inserting a temporal constraint, and is re-evaluated at the next iteration. The process enforces the occurrence of activities representing effects by posting temporal constraints which fix their start time to the current time. The effect of the constraints posted by these processes is that functionalities start when possible, are delayed until the preconditions hold, and their effects are imposed when necessary. This step also requires polynomial computation.

In our current implementation, all solvers monitor the network for new decision variables. Thus “re-planning” occurs by temporal propagation, resource or state variable scheduling, or operator application, depending on the situation. This mechanism is what enables dynamically posted goals, as in other temporal constraint-based continuous planners (McGann et al. 2008; Barreiro et al. 2012), but here we also deal with resources, sensor data and information constraints. We show an example of this behavior in the next Section.

5 Experiments

We now present a series of experiments aimed at describing the main features of our approach. All the experiments are related to the following elderly assistance scenario.

Sven lives in a smart home equipped with two service robots. The robots carry a pill dispenser and can fetch packages and mail delivered to the apartment. The apartment has 3 rooms: a living room (L), an entrance (E) and a kitchen (K) accessible through a controllable door. Sven has specified that he never wants two robots in the kitchen at the same time, as it is a cluttered environment. The robots require laser- or Kinect-based localization to navigate, but the Kinect should not be used in the kitchen for privacy concerns.

The motion of a robot is managed by the MoveFromTo functionality, whose localization requirement can be provided either by the robot’s laser-scanner or by its on-board Kinect. A resource of capacity one is associated to the kitchen, and all activities on state variables MoveFromTo and Location concerning the kitchen consume one unit of this resource. Another unit capacity resource is used as a mutex to make sure that Sven picks the mail and the pills from the robot sequentially.

In the following experiments, we describe the plans obtained given the following two goals: $g_1 = (\text{Pills}, \text{DELIVER}, I_1, \emptyset, \emptyset, \emptyset)$, which requires delivering pills to Sven who is in the kitchen; $g_2 = (\text{Post}, \text{DELIVER}, I_2, \emptyset, \emptyset, \emptyset)$, which requires delivering him the mail after picking it up at the entrance. The initial location of both robots is the living room, at their docking station. A third goal is always added, namely that a robot must return to the docking position after all other goals have been achieved. Due to space limitations, the final docking task is shown for one robot only.

In the first experiment, we provide only one robot, and we impose a deadline on g_1 . This constrains the robot to deliver the pills before fetching the mail (Figure 1, top). Both g_1 and g_2 are achieved by time $t = 110$. Relaxing the deadline

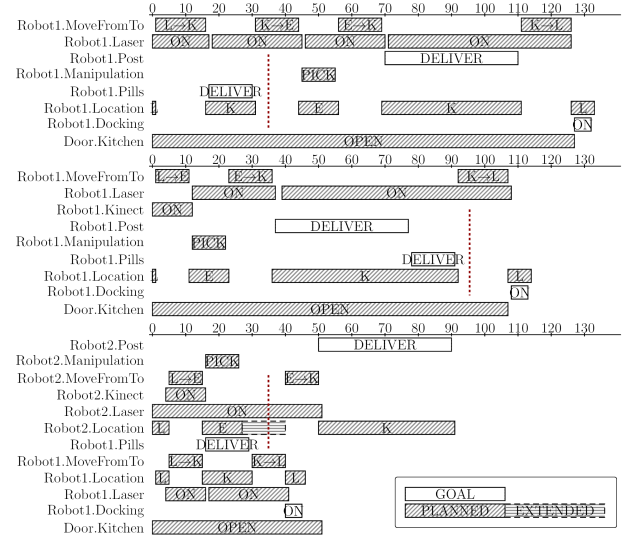


Figure 1: The three plans for achieving g_1 and g_2 (dotted lines indicate the deadline for g_1).

allows the robot to achieve g_1 and g_2 slightly earlier, as it has time to first go to the entrance and then to reach the kitchen, where it delivers both the mail and the pills within time $t = 90$ (see Figure 1, middle). In the second solution, the planner has exploited the path used for one sub-goal to achieve the other.

Considering both robots leads to achieving g_1 and g_2 even under the tighter deadline on g_1 , as mail fetching and pill delivery can be parallelized (Figure 1, bottom). The planner has resolved resource contention on the kitchen resource, as testified by the fact that $(\text{Robot2.Location}, E, I, \emptyset, \emptyset, \emptyset)$ has been extended until Robot1 has reached L.

5.1 Contingencies and Dynamic Goal Posting

The experiment shown next is executed in simulation with ROS/Gazebo² using two Willow Garage TurtleBots equipped with a laser range finder and a Kinect. The nominal durations of planned activities are modified during execution, and the constraint network is updated with constraints and activities representing sensor readings, as described in Section 4.

The experiment starts at time $t = 0$, when g_2 is posted. The resulting plan involves Robot2 (see Figure 2, top). At time $t = 20$, goal g_1 is posted. The planner accommodates this new goal, and execution proceeds as normal until $t = 70$, when a sudden delay appears: the arrival of the postman at the door will require more than the nominal duration of 20 (see Figure 2, middle), violating the deadline on g_1 . At this point, re-planning is triggered: functionalities currently in execution are retracted, and planning to achieve the two previously posted goals occurs. The new plan considers the situation reached through the previous plan, as well as sensor readings and execution status information (which have extended nominal durations of planned activities).

The plan resulting from re-planning involves the use of both robots, as one robot is stuck at the door waiting for

²<http://www.ros.org/>.

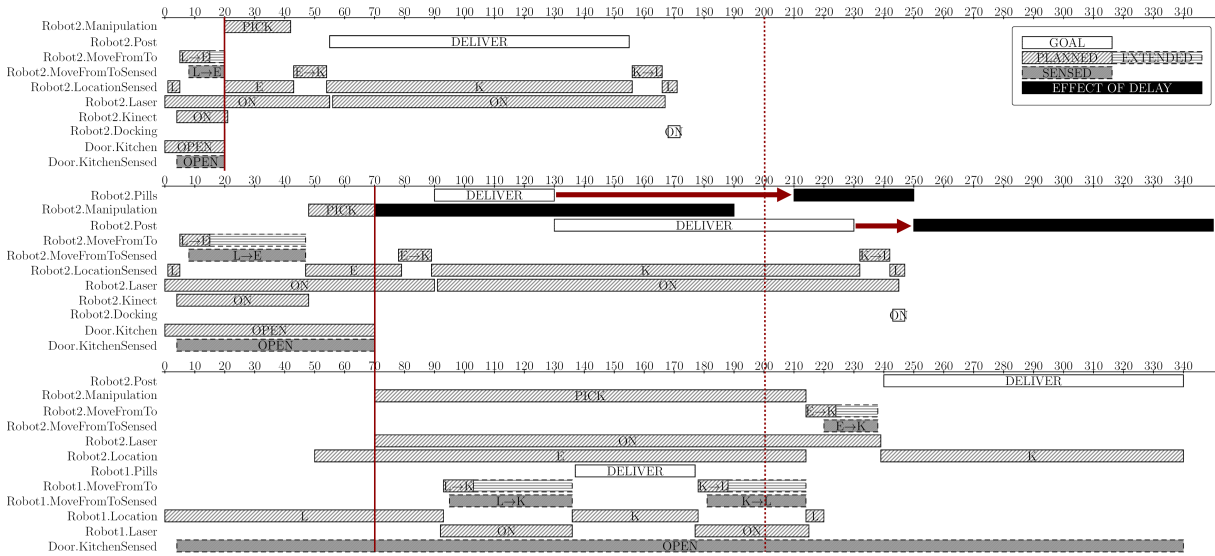


Figure 2: State of execution at times $t = 20$ (top), $t = 70$ (middle), $t = 340$ (bottom). The dotted line indicates the deadline for g_1 .

the arrival of the postman, while the other is free to deliver the pills. Figure 2 (bottom) shows the state of affairs at the end of execution ($t = 340$). The simulator output during significant moments in the execution of this experiment is shown in Figure 3. Videos of both the simulated run and one with real robots, together with the domain, are available at <http://aass.oru.se/~modo/AAAI-FSS-2013/>.

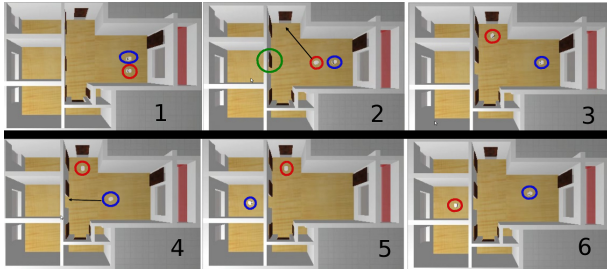


Figure 3: Simulator output (re-planning occurs at snapshot 3).

6 Conclusions

Our work is a first step towards building an approach to planning that can be used with real robotic systems. On-going work is addressing the issue of performance analysis and deployment in physically instantiated scenarios. In our future work we plan to study more sophisticated and efficient re-planning strategies, address the issue of including predicted/perceived human plans dynamically, consider spatial constraints, and take uncertainty into account.

References

Allen, J. 1984. Towards a general theory of action and time. *Artificial Intelligence* 23(2):123–154.

Barreiro, J.; Boyce, M.; Frank, J.; Iatauro, M.; Kichkaylo, T.; Morris, P.; Smith, T.; and Do, M. 2012. EUROPA: A platform for AI planning. In *Proc of ICAPS-ICKEPS*.

Cesta, A.; Oddi, A.; and Smith, S. F. 2002. A constraint-based method for project scheduling with time windows. *Journal of Heuristics* 8(1):109–136.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1-3):61–95.

Doherty, P.; Kvarnström, J.; and Heintz, F. 2009. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Autonomous Agents and Multi-Agent Systems* 19(3):332–377.

Fikes, R., and Nilsson, N. 1972. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3):189–208.

Ghallab, M., and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. In *AIPS*, 61–67.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Knight, S.; Rabideau, G.; Chien, S.; Engelhardt, B.; and Sherwood, R. 2001. Casper: Space exploration through continuous planning. *Intelligent Systems* 16(5):70–75.

Lundh, R.; Karlsson, L.; and Saffiotti, A. 2008. Autonomous functional configuration of a network robot system. *Robotics and Autonomous Systems* 56(10):819–830.

McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2008. A deliberative architecture for auv control. In *ICRA*, 1049–1054.

Parker, L., and Tang, F. 2006. Building multirobot coalitions through automated task solution synthesis. *Proc of the IEEE* 94(7):1289–1305.

Parker, L. 1998. ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Trans on Robotics and Automation* 14(2):220–240.

Saffiotti, A.; Broxvall, M.; Gritti, M.; LeBlanc, K.; Lundh, R.; Rashid, J.; Seo, B.; and Cho, Y. 2008. The PEIS-ecology project: vision and results. In *IROS*, 2329–2335.