Coordinating Agile Systems Through The Model-based Execution of Temporal Plans*

Thomas Léauté and **Brian Williams**
MIT Computer Science and Artificial Intelligence Laboratory (CSAIL)
Bldg. 32-G273, 77 Massachusetts Ave., Cambridge, MA 02139
{thleaute, williams}@mit.edu

Abstract

Agile autonomous systems are emerging, such as unmanned aerial vehicles (UAVs), that must robustly perform tightly coordinated time-critical missions; for example, military surveillance or search-and-rescue scenarios. In the space domain, execution of temporally flexible plans has provided an enabler for achieving the desired coordination and robustness. We address the challenge of extending plan execution to non-holonomic systems that are controlled indirectly through the setting of continuous state variables.

Our solution is a novel model-based executive that takes as input a temporally flexible *state plan*, specifying intended state evolutions, and dynamically generates an optimal control sequence. To achieve optimality and safety, the executive plans into the future, framing planning as a disjunctive programming problem. To achieve robustness to disturbances and tractability, planning is folded within a receding horizon, continuous planning framework. Key to performance is a problem reduction method based on constraint pruning. We benchmark performance through a suite of UAV scenarios using a hardware-in-the-loop testbed.

Introduction

Autonomous control of dynamic systems has application in a wide variety of fields, from managing a team of agile unmanned aerial vehicles (UAVs) for fire-fighting missions, to controlling a Mars life support system. The control of such systems is challenging for several reasons. First, they are non-holonomic systems, which means they are underactuated (not all state variables are directly controllable); second, their models involve continuous dynamics described by differential equations; third, controlling these systems usually requires tight synchronization; and fourth, the controller must be optimal and robust to disturbances.

An autonomous controller for agile systems must, therefore, provide three capabilities: 1) to handle tight coordination, the system should execute a temporal plan specifying time coordination constraints. 2) To deal with the underactuated nature of the system, it should elevate the interaction with the system under control (or *plant*) to the level at which the human operator is able to robustly program the plant in terms of desired state evolution, including state variables that are not directly controllable. 3) To deal with the non-holonomic dynamics of the plant, the intended state evolution must be specified in a temporally flexible manner, allowing robust control over the system.

Previous work in model-based programming introduced a *model-based executive*, called *Titan* (Williams 2003), that elevates the level of interaction between human operators and hidden-state, non-holonomic systems, by allowing the operator to specify the behavior to be executed in terms of intended plant state evolution, instead of specific command sequences. The executive uses models of the plant to map the desired state evolution to a sequence of commands driving the plant through the specified states. However, Titan focuses on reactive control of discrete-event systems, and does not handle temporally flexible constraints.

Work on dispatchable execution (Vidal & Ghallab 1996; Morris, Muscettola, & Tsamardinos 1998; Tsamardinos, Pollack, & Ramakrishnan 2003) provides a framework for robust scheduling and execution of temporally flexible plans. This framework uses methods based on distance graphs to both tighten time constraints in the plan, in order to guarantee dispatchability, and propagate occurrence of events during plan execution. However, this work was applied to discrete, directly controllable, loosely coupled systems, and, therefore, must be extended to non-holonomic plants.

Previous work on continuous planning and execution (Ambros-Ingerson & Steel 1988; Wilkins & Myer 1995; Chien *et al.* 2000) also provides methods to achieve robustness, by interleaving planning and execution, allowing on-the-fly replanning and adaptation to disturbances. These methods, inspired from model predictive control (*MPC*) (Propoi 1963; Richalet *et al.* 1976), involve planning and scheduling iteratively over short horizons, while revising the plan when necessary during execution. This work, however, needs to be extended to deal with temporally flexible plans and non-holonomic systems with continuous dynamics.

We propose a model-based executive that unifies the three previous approaches and enables coordinated control of agile systems, through model-based execution of *temporally flexible state plans*. Our approach is novel with respect to three aspects. First, we provide a general method for encod-

*This research is supported in part by The Boeing Company under contract MIT-BA-GTA-1, and by the Air Force Research Lab award under contract F33615-01-C-1850

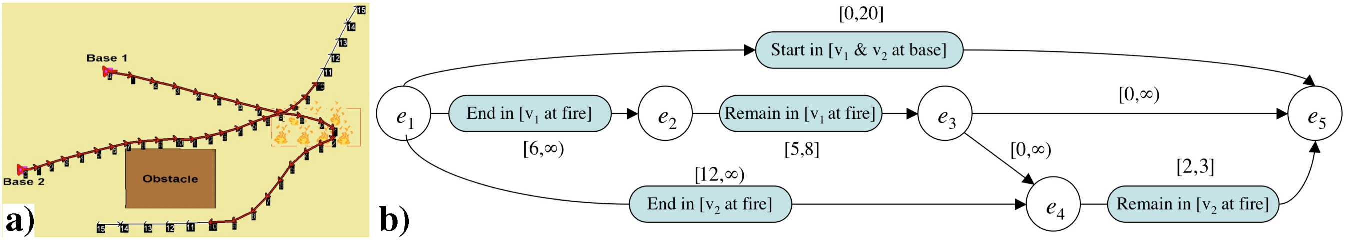

Figure 1: **a)** Map of the terrain for the fire-fighting example; **b)** Corresponding temporally flexible state plan.

ing both the temporal state plan and the dynamics of the system as a mixed discrete-continuous mathematical program. Solving this program provides time-optimal trajectories in the plant state space that satisfy the system dynamics and the state plan. Second, to achieve efficiency and robustness, we apply MPC for planning of control trajectories, in the context of continuous temporal plan execution for non-holonomic dynamical systems. MPC allows us to achieve tractability, by reasoning over a limited receding horizon. Third, in order to further reduce the complexity of the program and solve it in real time, we introduce pruning policies that enable us to ignore some of the constraints in the state plan outside the current planning horizon.

Problem Statement

Given a dynamic system (a *plant*) described by a *plant model*, and given a *temporally flexible state plan*, specifying the desired evolution of the plant state over time, the *continuous model-based execution* (*CMEx*) problem consists in designing a control sequence that produces a plant state evolution that is consistent with the state plan. In this section we present a formal definition of the CMEx problem.

Multiple-UAV Fire-fighting Example

This paragraph introduces the multiple-UAV fire-fighting example used in this paper. In this example, the plant consists of two fixed-wing UAVs, whose state variables are their 2-D Cartesian positions and velocities. The vehicles evolve in an environment (Fig. 1a) involving a reported fire that the team is assigned to extinguish. To do so, they must navigate around unsafe regions (e.g. obstacles) and drop water on the fire. They must also take pictures after the fire has been extinguished, in order to assess the damage. An English description for the mission's state plan is:

> *Vehicles v_1 and v_2 must start at their respective base stations. v_1 (a water tanker UAV) must reach the fire region and remain there for 5 to 8 time units, while it drops water over the fire. v_2 (a reconnaissance UAV) must reach the fire region after v_1 is done dropping water and must remain there for 2 to 3 time units, in order to take pictures of the damage. The overall plan execution must last no longer than 20 time units.*

Definition of a Plant Model

A *plant model* $\mathcal{M} = \langle \mathbf{s}, S, \mathbf{u}, \Omega, \mathcal{SE} \rangle$ consists of a vector $\mathbf{s}(t)$ of *state variables*, taking on values from the *state space* $S \subset \mathbb{R}^n$, a vector $\mathbf{u}(t)$ of *input variables*, taking on values from the *context* $\Omega \subset \mathbb{R}^m$, and a set $\mathcal{SE}$ of *state equations* over **u**, **s** and its time derivatives, describing the plant behavior with time. S and Ω impose linear safety constraints on **s** and **u**.

In our multiple-UAV example, **s** is the vector of 2-D coordinates of the UAV positions and velocities, and **u** is the acceleration coordinates. $\mathcal{SE}$ is the set of equations describing the kinematics of the UAVs. The unsafe regions in S correspond to obstacles and bounds on nominal velocities, and the unsafe regions in Ω to bounds on accelerations.

Definition of a Temporally Flexible State Plan

A *temporally flexible state plan* $P = \langle \mathcal{E}, \mathcal{C}, \mathcal{A} \rangle$ specifies a desired evolution of the plant state, and is defined by a set $\mathcal{E}$ of *events*, a set $\mathcal{C}$ of *coordination constraints*, imposing temporal constraints between events, and a set $\mathcal{A}$ of *activities*, imposing constraints on the plant state evolution. A coordination constraint $c = \langle e_1, e_2, \Delta T^{min}_{e_1 \to e_2}, \Delta T^{max}_{e_1 \to e_2} \rangle$ constrains the distance from event e_1 to event e_2 to be in $[\Delta T^{min}_{e_1 \to e_2}, \Delta T^{max}_{e_1 \to e_2}] \subset [0, +\infty]$. An *activity* $a = \langle e_1, e_2, c_S \rangle$ has an associated *start event* e_1 and an *end event* e_2. Given an assignment $T : \mathcal{E} \mapsto \mathbb{R}$ of times to all events in P (a *schedule*), c_S is a *state constraint* that can take on one of the following forms, where D_S, D_E, $D_\forall$ and $D_\exists$ are domains of S described by linear constraints on the state variables:

1. *Start in state region* D_S: $\mathbf{s}(T(e_1)) \in D_S$;
2. *End in state region* D_E: $\mathbf{s}(T(e_2)) \in D_E$;
3. *Remain in state region* $D_\forall$: $\forall t \in [T(e_1), T(e_2)], \mathbf{s}(t) \in D_\forall$;
4. *Go by state region* $D_\exists$: $\exists t \in [T(e_1), T(e_2)], \mathbf{s}(t) \in D_\exists$.

We illustrate a state plan diagrammatically by an acyclic directed graph in which events are represented by nodes, coordination constraints by arcs, labeled by their corresponding time bounds, and activities by arcs labeled with associated state constraints. The state plan for the the multiple-UAV fire-fighting mission example is shown in Fig. 1b.

Definition of the CMEx Problem

Schedule T for state plan P is *temporally consistent* if it satisfies all $c \in \mathcal{C}$. Given an activity $a = \langle e_1, e_2, c_S \rangle$ and a schedule T, a state sequence $\mathbf{S} = \langle \mathbf{s}_0 \ldots \mathbf{s}_t \rangle$ *satisfies activity a* if it satisfies c_S. **S** then *satisfies state plan P* if there exists a temporally consistent schedule such that **S** satisfies

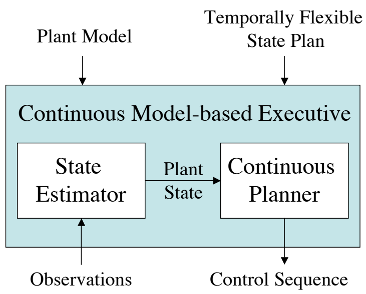

Figure 2: Continuous model-based executive architecture

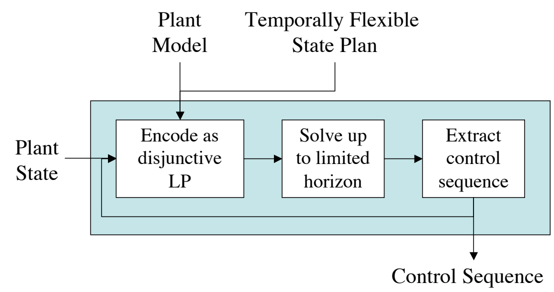

Figure 3: Receding horizon continuous planner

every activity in $\mathcal{A}$. Similarly, given a plant model $\mathcal{M}$ and initial state $\mathbf{s}_0$, a control sequence $\mathbf{U} = \langle \mathbf{u}_0 \ldots \mathbf{u}_t \rangle$ *satisfies* P if it generates a state sequence that satisfies P. $\mathbf{U}$ is *optimal* if it satisfies P while minimizing an *objective function* $F(\mathbf{U}, \mathbf{S}, T)$. A common objective is to minimize the scheduled time $T(e_E)$ for the end event e_E of P.

Given an initial state $\mathbf{s}_0$, a plant model $\mathcal{M}$ and state plan P, the CMEx problem consists of generating, for each $t > 0$, a control action $\mathbf{u}_t$ from a control sequence $\langle \mathbf{u}_0 \ldots \mathbf{u}_{t-1} \rangle$ and its corresponding state sequence $\langle \mathbf{s}_0 \ldots \mathbf{s}_t \rangle$, such that $\langle \mathbf{u}_0 \ldots \mathbf{u}_t \rangle$ is optimal. A corresponding *continuous model-based executive* consists of a *state estimator* and a *continuous planner* (Fig. 2). The continuous planner takes in a state plan, and generates optimal control sequences, based on the plant model, and state sequences provided by the state estimator. The estimator reasons on sensor observations and on the plant model in order to continuously track the state of the plant. Previous work on hybrid estimation (Hofbaur & Williams 2004) provides a framework for this state estimator; in this paper, we focus on presenting an algorithm for the continuous planner.

Overall Approach

Previous model-based executives, such as Titan, focus on reactively controlling discrete-event systems (Williams 2003). This approach is not applicable to temporal plan execution of systems with continuous dynamics; our continuous model-based executive uses a different approach that consists of planning into the future, in order to perform optimal, safe execution of temporal plans. However, solving the whole CMEx problem over an infinite horizon would present two major challenges. First, the problem is intractable in the case of long-duration missions. Second, it would require perfect knowledge of the state plan and the environment beforehand; this assumption does not always hold in real-life applications such as our fire-fighting scenario, in which the position of the fire might precisely be known only once the UAVs are close enough to the fire to localize it. Furthermore, the executive must be able to compensate possible drift due to approximations or errors in the plant model.

Receding Horizon CMEx

Model Predictive Control (*MPC*), also called *Receding Horizon Control*, is a method introduced in (Propoi 1963; Richalet *et al.* 1976) that tackles these two challenges in the context of low-level control of systems with continuous dynamics. MPC solves the control problem up to a limited *planning horizon*, and re-solves it when it reaches a shorter *execution horizon*. This method makes the problem tractable by restricting it to a small planning window; it also allows for on-line, robust adaptation to disturbances.

In this paper, we extend MPC to continuous model-based execution of temporal plans by introducing a *receding horizon continuous model-based executive*. We formally define receding horizon CMEx as follows. Given a state plan P, a plant model $\mathcal{M}$, and an initial state $\mathbf{s}(t_0)$, *single-stage, limited horizon CMEx* consists of generating an optimal control sequence $\langle \mathbf{u}_{t_0} \ldots \mathbf{u}_{t_0+N_t} \rangle$ for P, where N_t is the planning horizon. The *receding horizon CMEx* problem consists of iteratively solving single-stage, limited horizon CMEx for successive initial states $\mathbf{s}(t_0 + i \cdot n_t)$ with $i = 0, 1, \ldots$, where $n_t \leq N_t$ is the execution horizon. The architecture for our model-based executive is presented in Fig. 3.

Disjunctive Linear Programming Formulation

As introduced in Fig. 3, we solve each single-stage limited horizon CMEx problem by encoding it as a *disjunctive linear program* (*DLP*) (Balas 1979). A DLP is an optimization problem with respect to a linear cost function over decision variables, subject to constraints that can be written as logical combinations of linear inequalities (Eq. (1)). In this paper, we solve DLPs by reformulating them as Mixed-Integer Linear Programs. Other current work addresses directly solving DLPs (Krishnan 2004).

$$\begin{array}{c} \min f(\mathbf{x}) \\ subject\ to : \bigwedge_i \bigvee_j g_{i,j}(\mathbf{x}) \leq 0 \end{array} \tag{1}$$

Any arbitrary propositional logic formula whose propositions are linear inequalities is reducible to a DLP. Hence, in this paper, we expose formulae in propositional form.

Single-stage Limited Horizon CMEx as a DLP

We now present how we encode single-stage limited horizon CMEx as a DLP. Our innovation is the encoding of the state plan as a goal specification for the plant.

State Plan Encodings

In the following paragraphs, we present the encodings for the state plan. A range of objective functions are possible, the most common being to minimize completion time. To encode this, for every event $e \in \mathcal{E}$, we add to the DLP cost function, the time $T(e)$ at which e is scheduled.

Temporal constraints between events: Eq. (2) encodes a temporal constraint between two events e_S and e_E. For example, in Fig. 1b, events e_1 and e_5 must be distant from each other by a least 0 and at most 20 time units.

$$\Delta T^{\min}_{e_S \to e_E} \leq T(e_E) - T(e_S) \leq \Delta T^{\max}_{e_S \to e_E} \qquad (2)$$

State activity constraints: Activities are of the following types: *start in*, *end in*, *remain in* and *go by*. *Start in* and *go by* are derivable easily from the primitives *remain in* and *end in*. We present the encodings for these two primitives below. In each case, we assume that the domains D_E and $D_\forall$ are unions of polyhedra (Eq. (6)), so that $\mathbf{s}_t \in D_E$ and $\mathbf{s}_t \in D_\forall$ can be expressed as DLP constraints similar to (Eq. (7)).

***Remain in* activity:** Eq. (3) presents the encoding for the *remain in state* $D_\forall$ activity between events e_S and e_E. This imposes $\mathbf{s} \in D_\forall$ for all time steps between $T(e_S)$ and $T(e_E)$. Our example imposes the constraint "Remain in state [v_1 in fire region]", which means that v_1 must be in the fire region between e_2 and e_3 while it is dropping water. $T_0 \in \mathbb{R}$ denotes the initial time instant of index $t = 0$.

$$\bigwedge_{t=0\ldots N_t} \left\{ \begin{array}{rc} & T(e_S) \leq T_0 + t \cdot \Delta T \\ \wedge & T(e_E) \geq T_0 + t \cdot \Delta T \end{array} \right\} \Rightarrow \mathbf{s}_t \in D_\forall \qquad (3)$$

***End in* activity:** Consider a *end in* activity imposing $\mathbf{s} \in D_\forall$ at the time $T(e_E)$ when event e_E is scheduled. An example in our fire-fighting scenario is the "End in state [v_2 in fire region]" constraint imposing v_2 to be in the fire region at event e_4. The general encoding is presented in Eq. (4), which translates to the fact that, either there exists a time instant of index t in the planning window that is ΔT-close to $T(e_E)$ and for which $\mathbf{s}_t \in D_\forall$, or event e_E must be scheduled outside of the current planning window .

$$\begin{array}{cc} \bigvee_{t=0\ldots N_t} & \left\{ \begin{array}{rc} & T(e_E) \geq T_0 + (t - \frac{1}{2})\Delta T \\ \wedge & T(e_E) \leq T_0 + (t + \frac{1}{2})\Delta T \\ \wedge & \mathbf{s}_t \in D_\forall \end{array} \right\} \\ \vee & T(e_E) \leq T_0 - \frac{\Delta T}{2} \\ \vee & T(e_E) \geq T_0 + (N_t + \frac{1}{2})\Delta T \end{array} \qquad (4)$$

Guidance heuristic for *End in* activities: During execution of an "End in state region D_E" activity, the end event may be scheduled beyond the current horizon. In this case, the model-based executive constructs a *heuristic*, in order to guide the trajectory towards D_E. For this purpose, an estimate of the "distance" to D_E from the end of the current partial state trajectory is added to the DLP cost function, so that the partial trajectory ends as "close" to D_E as possible. In the case of the multiple-UAV fire-fighting scenario, this "distance" is an estimate of the time needed to go from the end of the current trajectory to the goal region D_E.

The heuristic is formally defined as a function $h_{D_E} : \mathcal{S} \mapsto \mathbb{R}$, where $\mathcal{S} = \{S_i \subset S\}$ is a finite partition of S such that $D_E = S_{i_{D_E}} \in \mathcal{S}$. Given $S_i \in \mathcal{S}$, $h_{D_E}(S_i)$ is an estimate of the cost to go from S_i to D_E. In the fire-fighting example, $\mathcal{S}$ is a grid map in which each grid cell S_i corresponds to a hypercube centered on a state vector $\mathbf{s}_i$, and $h_{D_E}(S_i)$ is an estimate of the time necessary to go from state $\mathbf{s}_i$ to the goal state $\mathbf{s}_{i_{D_E}}$. Similar to (Bellingham, Richards, & How 2002), we compute h_{D_E} by constructing a visibility graph based on the unsafe regions of the $\langle x, y\rangle$ state space, and by computing, for every i, the cost to go from $\langle x_i, y_i\rangle \in S_i$ to the goal state $\langle x_{i_{D_E}}, y_{i_{D_E}}\rangle \in D_E$.

Eq. (5) presents the constraint, for a given "End in state region D_E" activity a, starting at event e_S and ending at event e_E. This encodes the fact that, if a is scheduled to start within the execution horizon but end beyond, then the executive must choose a region $S_i \in \mathcal{S}$ so that the partial state trajectory ends in S_i, and the value h of the heuristic at S_i is minimized (by adding h to the DLP cost function).

$$\begin{array}{l} \left\{ \begin{array}{rc} & T(e_S) < T_0 + n_t \cdot \Delta T \\ \wedge & T(e_E) \geq T_0 + n_t \cdot \Delta T \end{array} \right\} \\ \Rightarrow \bigvee_{S_i \in \mathcal{S}} \left\{ \begin{array}{rc} & h = h_{D_E}(S_i) \\ \wedge & \mathbf{s}_{n_t} \in S_i \end{array} \right\} \end{array} \qquad (5)$$

Eq. (5) can be simplified by reducing $\mathcal{S}$ to a subset $\tilde{\mathcal{S}} \subset \mathcal{S}$ that excludes all the S_i unreachable within the horizon. For instance, in the multiple-UAV example, the maximum velocity constraints allow us to ignore the S_i that are not reachable by the UAVs within the execution horizon. We present in a later section how $\mathcal{M}$ allows us to determine, in the general case, when a region of the state space is unreachable.

Plant Model Encodings

Recall that a plant model $\mathcal{M}$ consists of a state space S and a context Ω imposing linear constraints on the variables, and a set of state equations $\mathcal{SE}$. We represent unsafe regions in S and Ω by unions of polyhedra, where a polyhedron $\mathcal{P}_S$ of S is defined in Eq. (6). Polyhedra of Ω are defined similarly.

$$\mathcal{P}_S = \left\{ \mathbf{s} \in \mathbb{R}^n \mid \mathbf{a}_i^T \mathbf{s} \leq b_i \,, i = 1 \ldots n_{\mathcal{P}_S} \right\} \qquad (6)$$

The corresponding encoding (Eq. (7)) constrains $\mathbf{s}_t$ to be outside of $\mathcal{P}_S$ for all t. In the UAV example, this corresponds to the constraint encoding obstacle collision avoidance.

$$\bigwedge_{t=1\ldots N_t} \bigvee_{i=1\ldots n_{\mathcal{P}_S}} \mathbf{a}_i^T \mathbf{s}_t \geq b_i \qquad (7)$$

The state equations in $\mathcal{SE}$ are given in DLP form in Eq. (8), with the time increment ΔT assumed small with respect to the plant dynamics. In our fixed-wing UAV example, we use the zero-order hold time discretization model from (Kuwata 2003).

$$\mathbf{s}_{t+1} = \mathbf{A}\mathbf{s}_t + \mathbf{B}\mathbf{u}_t \qquad (8)$$

In this section, we presented how we designed optimal control sequences that satisfy the state plan, by formulating the problem as a DLP using an MPC framework. We now present how we simplify the DLP to solve it in real time.

Constraint Pruning Policies

Recall that our executive solves the CMEx problem by encoding it as a DLP and iteratively solving it over small planning windows. The ability of the executive to look into the future is limited by the number of variables and constraints in the DLP. In the next section, we introduce novel *pruning policies* that dramatically reduce the number of constraints.

Plant Model Constraint Pruning

Recall that $\mathcal{M}$ defines unsafe regions in S using polyhedra (Eq. (6)). The DLP constraint for a polyhedron $\mathcal{P}_S$ (Eq. (7)) can be pruned if $\mathcal{P}_S$ is unreachable from the current plant state $\mathbf{s}_0$, within the horizon N_t. That is, if the region R of all states reachable from $\mathbf{s}_0$ within N_t is disjunct from $\mathcal{P}_S$. R is formally defined in Eq. (9) and (10), with $R_0 = \{\mathbf{s}_0\}$.

$$\forall t = 0 \ldots N_t - 1, \quad R_{t+1} = \left\{ \mathbf{s}_{t+1} | \begin{array}{c} \mathbf{s}_{t+1} = \mathbf{A}\mathbf{s}_t + \mathbf{B}\mathbf{u}_t, \\ \mathbf{s}_t \in R_t, \mathbf{u}_t \in \Omega \end{array} \right\} \tag{9}$$

$$R = \bigcup_{t=0 \ldots N_t} R_t \tag{10}$$

Techniques have been developed in order to compute R (Tiwari 2003). In the UAV example, for a given vehicle, we use a simpler, sound but incomplete method, in which we approximate R by a circle centered on the vehicle and of radius $N_t \cdot \Delta T \cdot v^{\max}$, where $v^{\max}$ is the vehicle's maximum velocity.

State Plan Constraint Pruning

State plan constraints can be either temporal constraints between events, *remain in* constraints, *end in* constraints, or heuristic guidance constraints for *end in* activities. For each type, we now show that the problem of finding a policy is equivalent to that of foreseeing if an event could possibly be scheduled within the current horizon. This is solved by computing bounds $\langle T_e^{\min}, T_e^{\max} \rangle$ on $T(e)$, for every $e \in \mathcal{E}$. Given the execution times of past events, these bounds are computed from the bounds $\langle \Delta T_{\langle e,e' \rangle}^{\min}, \Delta T_{\langle e,e' \rangle}^{\max} \rangle$ on the distance between any pair of events $\langle e, e' \rangle$, obtained using the method in (Dechter, Meiri, & Pearl 1991). This involves running an all-pairs shortest path algorithm on the distance graph corresponding to P, which can be done offline.

Temporal constraint pruning: A temporal constraint between a pair of events $\langle e_S, e_E \rangle$ can be pruned if the time bounds on either event guarantee that the event will be scheduled outside of the current planning window (Alg. 1).

However, pruning some of the temporal constraints specified in the state plan can have two bad consequences. First, implicit temporal constraints between two events that can be scheduled within the current planning window might no longer be enforced. Implicit temporal constraints are constraints that do not appear explicitly in the state plan, but rather result from several explicit temporal constraints. Second, the schedule might violate temporal constraints between events that remain to be scheduled, and events that have already been executed.

Alg. 1 Pruning policy for the temporal constraint between events e_S and e_E

if $T_{e_S}^{\max} < T_0$ **then**
  prune $\{e_S$ has already been executed$\}$
else if $T_{e_S}^{\min} > T_0 + N_t \cdot \Delta T$ **then**
  prune$\{e_S$ is out of reach within the current horizon$\}$
else if $T_{e_E}^{\max} < T_0$ **then**
  prune $\{e_E$ has already been executed$\}$
else if $T_{e_E}^{\min} > T_0 + N_t \cdot \Delta T$ **then**
  prune$\{e_E$ is out of reach within the current horizon$\}$
end if

To tackle the first issue aforementioned, rather than encoding only the temporal constraints that are mentioned in the state plan, we encode the temporal constraints between any pair of events $\langle e, e' \rangle$, using the temporal bounds $\langle \Delta T_{\langle e,e' \rangle}^{\min}, \Delta T_{\langle e,e' \rangle}^{\max} \rangle$ computed by the method in (Dechter, Meiri, & Pearl 1991). This way, no implicit temporal constraint is ignored, because all temporal constraints between events are explicitly encoded.

To address the second issue, we also encode the absolute temporal constraints on every event e: $T_e^{\min} \leq T(e) \leq T_e^{\max}$. The pruning policy for those constraints is presented in Alg. 2. The constraint can be pruned if e is guaranteed to be scheduled in the past (i.e. it has already been executed, line 1). It can also be pruned if e is guaranteed to be scheduled beyond the current planning horizon (line 3). In that case, e must be explicitly postponed (Alg. 3, lines 1 & 2) to make sure it will not be scheduled before T_0 at the next iteration (which would then correspond to scheduling e in the past before it has been executed). The change in $T_e^{\min}$ is then propagated to the other events (Alg. 3, line 3).

Alg. 2 Pruning policy for the absolute temporal constraint on an event e

1: **if** $T_e^{\max} < T_0$ **then**
2:   prune $\{e$ has already been executed$\}$
3: **else if** $T_e^{\min} > T_0 + N_t \cdot \Delta T$ **then**
4:   prune$\{e$ is out of reach within the current horizon$\}$
5:   POSTPONE(e)
6: **end if**

***Remain in* constraint pruning (Alg. 4):** Consider the constraint c_S on a "Remain in state region $D_\forall$" activity a, between events e_S and e_E (Eq. (3)). If e_E is guaranteed to be scheduled in the past (i.e. it has already occurred, line 1), then a has been completed and c_S can be pruned. Otherwise, if e_S has already occurred (line 3), then a is being executed and c_S must not be pruned. Otherwise, if a is guaranteed

Alg. 3 POSTPONE(e) routine to postpone an event e

1: $T_e^{\min} \leftarrow T_0 + N_t \cdot \Delta T$
2: add $T(e) \geq T_0 + N_t \cdot \Delta T$ to the DLP
3: **for all** events e' **do** {propagate to other events}
4: $\quad T_{e'}^{\min} \leftarrow \max(T_{e'}^{\min}, T_e^{\min} + \Delta T_{\langle e,e' \rangle}^{\min})$
5: **end for**

to start beyond the planning horizon (line 5), then c_S can be pruned. Conversely, if a is guaranteed to start within the planning horizon (line 7), then c_S must not be pruned.

Alg. 4 Pruning policy for a "Remain in state region $D_\forall$" activity starting at event e_S and ending at event e_E

1: **if** $T_{e_E}^{\max} < T_0$ **then**
2: $\quad$ prune {activity is completed}
3: **else if** $T_{e_S}^{\max} < T_0$ **then**
4: $\quad$ do not prune{activity is being executed}
5: **else if** $T_{e_S}^{\min} > T_0 + N_t \cdot \Delta T$ **then**
6: $\quad$ prune {activity will start beyond N_t}
7: **else if** $T_{e_S}^{\max} < T_0 + N_t \cdot \Delta T$ **then**
8: $\quad$ do not prune {activity will start within N_t}
9: **else if** $R \cap D_\forall = \emptyset$ **then**
10: $\quad$ prune; POSTPONE(e_S)
11: **end if**

Otherwise, the time bounds on $T(e_S)$ and $T(e_E)$ provide no guarantee, but we can still use $\mathcal{M}$ to try to prune the constraint: if $\mathcal{M}$ guarantees that $D_\forall$ is unreachable within the planning horizon, then c_S can be pruned (line 9; refer to Eq. (9) & (10) for the definition of R). Similarly to Alg. 2, e_S must then be explicitly postponed.

***End in* constraint pruning (Alg. 5):** Consider a constraint c_S on an "End in state region D_E" activity ending at event e_E (Eq. (4)). If e_E is guaranteed to be scheduled in the past (i.e., it has already occurred, line 1), then c_S can be pruned. Otherwise, if the value of $T_{e_E}^{\max}$ guarantees that e_E will be scheduled within the planning horizon (line 3), then c_S must not be pruned. Conversely, it can be pruned if $T_{e_E}^{\min}$ guarantees that e_E will be scheduled beyond the planning horizon (line 5). Finally, c_S can also be pruned if the plant model guarantees that D_E is unreachable within the planning horizon from the current plant state (line 7). Similarly to Alg. 2, e_E must then be explicitly postponed.

Guidance constraint pruning: The heuristic guidance constraint for an *end in* activity a between events e_S and e_E (Eq. (5)) can be pruned if a is guaranteed either to end within the execution horizon ($T_{e_E}^{\max} < T_0 + n_t \cdot \Delta T$) or to start beyond ($T_{e_S}^{\min} > T_0 + n_t \cdot \Delta T$).

In the next section, we show that our model-based executive is able to design optimal control sequences in real time.

Alg. 5 Pruning policy for a "End in state region D_E" activity ending at event e_E

1: **if** $T_{e_E}^{\max} < T_0$ **then**
2: $\quad$ prune {e_E has already occurred}
3: **else if** $T_{e_E}^{\max} \leq T_0 + N_t \cdot \Delta T$ **then**
4: $\quad$ do not prune {e_E will be scheduled within N_t}
5: **else if** $T_{e_E}^{\min} > T_0 + N_t \cdot \Delta T$ **then**
6: $\quad$ prune {e_E will be scheduled beyond N_t}
7: **else if** $R \cap D_E = \emptyset$ **then**
8: $\quad$ prune; POSTPONE(e_E)
9: **end if**

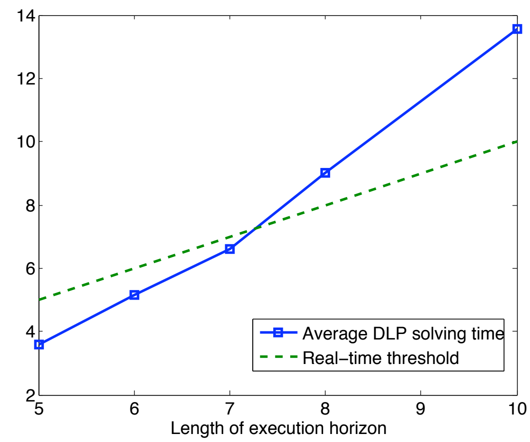

Figure 4: Performance of the model-based executive.

Results and Discussion

The model-based executive has been implemented in C++, using Ilog CPLEX to solve the DLPs. It has been demonstrated on a range of fire-fighting scenarios on a hardware-in-the-loop testbed, comprised of CloudCap autopilots controlling a set of fixed-wing UAVs. This offers a precise assessment of real-time performance on UAVs. Fig. 1a was obtained from a snapshot of the operator interface, and illustrates the trajectories corresponding to our example.

Fig. 4 presents an analysis of the performance of the executive on a more complex example, comprised of two vehicles, two obstacles, and 26 activities, for a total execution time of about 1300s. These results were obtained on a 1.7GHz computer with 512MB RAM, by averaging over the whole plan execution, and over 5 different runs with random initial conditions. At each iteration, the computation was cut short if and when it passed 200s.

The x axis corresponds to the length of the execution horizon, $n_t \cdot \Delta T$, in seconds. For these results, we maintained a planning buffer of $N_t \cdot \Delta T - n_t \cdot \Delta T = 10s$ (where $N_t \cdot \Delta T$ is the length of the planning horizon). The full line represents the average time in seconds required by CPLEX to solve the DLP at each iteration, while the dotted line is the line $y = x$, corresponding to the real-time threshold.

Fig. 4 shows that, below the value $x \simeq 7.3s$, the model-based executive is able to compute optimal control sequences in real time (the average DLP solving time is below the length of the execution horizon). For longer planning horizons corresponding to values of x above $7.3s$, CPLEX is unable to find optimal solutions to the DLPs before the executive is required to replan (average solving time greater than the execution horizon). Note that in that case, since CPLEX runs as an anytime algorithm, we can still interrupt it and use the best solution found so far to generate sub-optimal control sequences.

Note also that the number of variables in the DLP is linear in the length of the planning horizon; therefore, the complexity of the DLP is worst-case exponential in the length of the horizon. In Fig. 4, however, the relationship appears to be linear. This can be explained by the fact that the DLP is very sparse, since no constraint in the corresponding MILP involves more than three or four variables.

Future work includes carrying out more experiments in order to benchmark the performance of the pruning policies presented in this paper.

Conclusion

In this paper, we have presented a continuous model-based executive that is able to robustly execute temporal plans for agile, non-holonomic systems with continuous dynamics. In order to deal with the under-actuated nature of the plant and to provide robustness, the model-based executive reasons on temporally flexible state plans, specifying the intended plant state evolution. The use of pruning policies enables the executive to design optimal control sequences in real time, which was demonstrated on a hardware-in-the-loop testbed in the context of multiple-UAV fire-fighting scenarios. Our approach is broadly applicable to other dynamic systems, such as chemical plants or Mars life support systems.

References

Ambros-Ingerson, J. A., and Steel, S. 1988. Integrating planning, execution and monitoring. In *Proc. AAAI*.

Balas, E. 1979. Disjunctive programming. *Annals of Discrete Mathematics* 5:3–51.

Bellingham, J.; Richards, A.; and How, J. 2002. Receding horizon control of autonomous aerial vehicles. In *ACC*.

Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 2000. Using iterative repair to improve responsiveness of planning and scheduling. In *Proc. AIPS*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence Journal*.

Hofbaur, M. W., and Williams, B. C. 2004. Hybrid estimation of complex systems. *IEEE SMC - Part B: Cybernetics*.

Krishnan, R. 2004. Solving hybrid decision-control problems through conflict-directed branch and bound. Master's thesis, MIT.

Kuwata, Y. 2003. Real-time trajectory design for unmanned aerial vehicles using receding horizon control. Master's thesis, MIT.

Morris, P.; Muscettola, N.; and Tsamardinos, I. 1998. Reformulating temporal plans for efficient execution. In *Proc. KRR*, 444–452.

Propoi, A. 1963. Use of linear programming methods for synthesizing sampled-data automatic systems. *Automation and Remote Control* 24(7):837–844.

Richalet, J.; Rault, A.; Testud, J.; and Papon, J. 1976. Algorithmic control of industrial processes. In *IFAC Sym. Id. & Syst. Param. Est.*, 1119–1167.

Tiwari, A. 2003. Approximate reachability for linear systems. In *Proceedings of HSCC*, 514–525.

Tsamardinos, I.; Pollack, M. E.; and Ramakrishnan, S. 2003. Assessing the probability of legal execution of plans with temporal uncertainty. In *Proc. ICAPS*.

Vidal, T., and Ghallab, M. 1996. Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *Proc. ECAI*, 48–52.

Wilkins, D. E., and Myer, K. L. 1995. A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation* 5(6):731–761.

Williams, B. C. 2003. Model-based programming of intelligent embedded systems and robotic space explorers. In *Proc. IEEE: Modeling and Design of Embedded Software*.