Integrated Planning, Execution and Estimation for Robotic Exploration

Conor McGann*, **Frédéric Py, Kanna Rajan, Angel Garcia Olaya**†
Monterey Bay Aquarium Research Institute, Moss Landing, California
{*cmcgann, fpy, kanna.rajan, agolaya*}@mbari.org

Abstract

This paper uses Constraint-based Temporal Planning (CTP) techniques to integrate deliberation and reaction in a uniform representation for autonomous robot control. Further we inform planning with an onboard Hidden Markov-Model for environmental state estimation. We formulate a control structure that *partitions* an agent into a collection of coordinated control loops, with a recurring sense, plan, act cycle. Algorithms are presented for sharing state between controllers to ensure consistency and enable compositional control. The partitioned structure makes it practical to apply CTP for both deliberative and reactive behavior and promises a scalable and robust approach for control of real-world autonomous robots operating in dynamic environments while showing how estimation driven planning can make robots adaptive.

1 Introduction

The ocean plays a crucial role in the ecosystem of our planet. This vast, hostile, unstructured and unpredictable environment has proven resistant to extensive scientific study. Its depths are largely inaccessible to humans, and impervious to surface and space based observation techniques. Oceanographic ship-based measurements have recently been augmented by untethered robotic platforms such as Autonomous Underwater Vehicles (AUVs) [Yuh, 2000]. They carry sophisticated science payloads for measuring important water properties [Ryan *et al.*, 2005], as well as instruments for recording the morphology of the benthic environment with advanced sonar equipment [Thomas, 2006]. The extensive payload capacity and operational versatility of these vehicles offer a cost-effective alternative to current methods of oceanographic measurements.

The capabilities of AUVs and the challenges of ocean exploration provide a compelling domain to motivate and evaluate advances in robotics. Our research is focused on software techniques to enhance the robustness, adaptability, usability and utility of AUVs in the pursuit of important scientific questions in the deep ocean. Figure 1 is a composite image of our platform, the *Dorado* vehicle, which operates routinely to depths of 1500m and the mid-section Gulper water sampling instrument.

We have developed and deployed an onboard Adaptive Control System that integrates Planning and Probabilistic State Estimation in a hybrid Executive. Probabilistic State Estimation integrates a number of science observations to produce a likelihood that the vehicle sensors perceive a feature of interest. Onboard planning and execution enables adaptation of navigation and instrument control based on the probability of having detected such a phenomenon. It further enables goal-directed commanding within the context of projected mission state and allows for replanning for off-nominal situations and opportunistic science events.

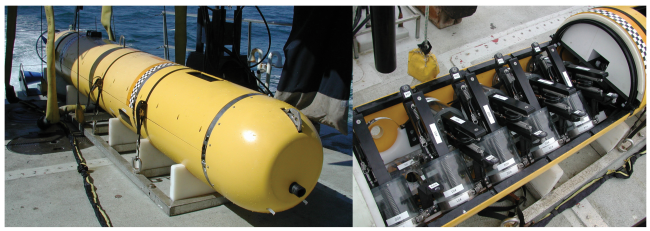

Fig. 1: MBARI's Dorado Autonomous Underwater Vehicle at sea (left) and its mid-body water sampler (right)

The novelty of this work is twofold. We integrate deliberation and reaction systeatically over different temporal and functional scopes within a single agent and a single model that covers the needs of high-level mission management, low-level navigation, instrument control, and detection of unstructured and poorly understood phenomena. Secondly, we break new ground in oceanography by allowing scientists to obtain samples precisely within a scientific feature of interest using an autonomous robot.

Our primary interests hertofore have been to track *dynamic* upper water-column features such as ocean fronts, Intermediate Nepheloid Layers (INLs; fluid sheets of suspended particulate matter that originate from the sea floor), and blooms (patches of high biological activity). Each of these phenomena can occur over a wide range of scales, from thousands of kilometers (fronts and blooms) down to tens/hundreds of meters, and each is characterized by strong spatio-temporal dependence. This dependence results in a high degree of unpredictability in the location and intensity of the phenomena, and this unpredictability is the primary motivation for using novel techniques to resolve the small-scale features of each of the phenomena.

The remainder of this paper is organized as follows. Sec-

tion 2 places this work in the context of other integrated planning and execution frameworks. Section **??** introduces key concepts and definitions. The core of the paper in section 4 presents algorithms for coordinated control and synchronization of state with complexity analysis while Section 5 shows how estimation using HMMs impact planning. Section 6 concludes with empirical data from experiments evaluating algorithm performance.

2 Related Work

The dominant approach for building agent control systems utilize a three-layered architecture [Gat, 1998], notable examples of which include IPEM [Ambros-Ingerson and Steel, 1988], ROGUE [Haigh and Veloso, 1998], the LAAS Architecture [Alami *et al.*, 1998], the Remote Agent Experiment [Muscettola *et al.*, 1998] & [Rajan *et al.*, 2000] and ASE [Chien *et al.*, 1999] (see [Knight *et al.*, 2001] for a survey). Scalability is of concern since the planning cycle in these approaches is monolithic often making fast reaction times impractical when necessary. Many of these systems also utilize very different techniques for specifying each layer in the architecture resulting in duplication of effort and a diffusion of knowledge. IDEA [Muscettola *et al.*, 2002] was the first agent control architecture utilizing a collection of controllers, each interleaving planning and execution in a common framework. However, IDEA provides no support for conflict resolution between agents, nor did it provide an efficient algorithm for integrating current state within a controller, relying instead on a possibly exponential planning algorithm. In our view, efficient synchronization of state in a partitioned structure is fundamental to making the approach effective in practice. Our work provides a formal basis for partitioning a complex control problem to achieve scalability and robustness and an approach for synchronization in polynomial time.

3 Key Concepts & Definitions

Autonomous robotic explorers must be proactive in the pursuit of goals and reactive to evolving environmental conditions. These concerns must be balanced over short and long term horizons to consider timeliness, safety and efficiency, presenting a substantial challenge for control system design. As an example, a planetary rover is often tasked with a set of objectives to be accomplished over the course of a mission. It may reasonably deliberate for many minutes if necessary; in contrast, instrument control decisions require faster reaction times but can be taken with a more myopic view of implications to the plan. This suggests that the control responsibilities of the executive can be *partitioned* to exploit differing requirements in terms of which state variables need to be considered together, over what time frames, and at what reaction rates. Such a partitioned structure could increase the complexity of integration however, especially if components are constructed with disparate technologies for program specification and/or execution.

To address this integration problem, we formally define an agent control structure as a composition of coordinated control loops with a recurring sense, plan, act (SPA) cycle. Representational primitives in our framework are based on the semantics of Constraint-based Temporal Plans [Frank and Jónsson, 2003] using unified declarative models. We manage the information flow within the partitioned structure to ensure consistency in order to direct the flow of goals and observations in a timely manner. The resulting control structure improves scalability since many details of each controller can be encapsulated within a single control loop. Furthermore, partitioning increases robustness since controller failure can be localized to enable graceful system degradation, making this an effective divide-and-conquer approach to the overall control problem.

We formally define a situated agent in a world $\mathcal{W}$ by:

- A set of **state variables**: $\mathcal{S} = \{s_1, \ldots, s_n\}$
- A **lifetime**: $\mathcal{H} = [0, \Pi)$ defining the interval of time in which the agent will be active. $\mathcal{H} \subseteq \mathbb{N}$.
- The **execution frontier**: $\tau \in \mathcal{H}$ is the elapsed execution time as perceived by the agent. This value increases as time advances in $\mathcal{W}$. The unit of time is called a *tick*. The agent observes and assumes world evolution at the tick rate; changes between ticks are ignored.

During its lifetime an agent observes the evolution of the world through $\mathcal{S}$ via timelines as the execution frontier advances [Finzi *et al.*, 2004].

Definition 1 *A timeline, L, is defined by:*

- *a* **state variable***: $s(L) \in \mathcal{S}$.*
- *a set of* **tokens** *assigned to this timeline: $\mathcal{T}(L)$. Each token $t \in \mathcal{T}(L)$ expresses a constraint on the value of s over some temporal extent. A token $p(start, end, \overrightarrow{x})$ indicates that the predicate p with its attributes $\overrightarrow{x}$ holds for the temporal domain $[start, end)$ where start and end are flexible temporal intervals.*
- *an ordered set $\mathcal{Q}(L) \subseteq \mathcal{T}(L)$ of tokens describing the evolution of the state variable over time.*

We use the notation, $L(t)$ to refer to the set of the tokens *ordered* in the timeline L that overlaps time t:

$$L(t) = \{a; a \in \mathcal{Q}(L) \wedge a.start \leq t < a.end\}$$

We introduce a primitive for control called a *reactor* with which the global control structure is composed. Coordination is based on an explicit division of authority for determining the value for each state variable among the set of reactors, *R*. A reactor may *own* or *use* one or more state variables. A state variable s is *owned* by one and only one reactor r ($owns(r, s)$) which has the unique authority to determine the value for that state variable as execution proceeds. Conversely, a reactor r *uses* a state variable s ($uses(r, s)$) if it needs to be notified of changes (*observations*) or it needs to request values for this state variable (*goals*). Coordination is also based on explicit information about the *latency* and temporal scope or *look-ahead* of deliberation of a reactor. This information is used to ensure information is shared as needed and no sooner.

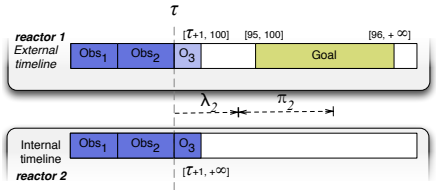

Fig. 2: Two reactors sharing one state variable. Observations from the past are consistent while projections in the future can differ to reflect each reactor's plan.

Definition 2 *As illustrated in Fig.2 a reactor r is a controller defined by:*

- *a* **latency***: λ_r is the maximum amount of time the reactor r can use for deliberation.*
- *a* **look-ahead***: $\pi_r \in [\lambda_r, \Pi]$ defines the temporal duration over which reactor r deliberates. When deliberation starts for reactor r, its planning horizon is:*

$$h_r = [\tau + \lambda_r, \tau + \lambda_r + \pi_r) \quad (1)$$

- *a set of* **internal** *timelines: $\mathcal{I}_r = \{I_1, \ldots, I_k\}$. Timelines in $\mathcal{I}_r$ refer to state variables reactor r* owns.
- *a set of* **external** *timelines: $\mathcal{E}_r = \{E_1, \ldots, E_l\}$. Timelines in $\mathcal{E}_r$ refer to state variables reactor r* uses. *$\varepsilon(s)$ defines all external timelines referring to a given state variable s as $\varepsilon(s) = \{E : E \in \cup_{r \in \mathcal{R}} \mathcal{E}_r, s(E) = s\}$*
- *a set of* **goal tokens***: $\mathcal{G}_r$. Goal tokens express constraints on the future values of a state variable* owned *by this reactor.*
- *a set of* **observation tokens***: $\mathcal{O}_r$. Observation tokens express present and past values of a state variable* ***used*** *by this reactor. They must be identical to the corresponding values of this state variable as described by its* owner.
- *a* **model***: $\mathcal{M}_r$. The model defines the rules governing the interactions between values on and across timelines.*

We consider a special set of reactors, $\mathcal{R}_w$, which includes all reactors r that have no *external* dependency (i.e $\mathcal{E}_r = \emptyset$). In practice, such primitive reactors encapsulate the exogenous state variables of the agent. We further define the model for the agent, $\mathcal{M}_w$, as the union of all models of each individual reactor: $\mathcal{M}_w = \bigcup_{r \in \mathcal{R}} \mathcal{M}_r$

Fig.3 for instance, shows an agent with four reactors. A Mission Manager provides high-level directives to satisfy the scientific and operational goals of the mission: its temporal scope is the entire mission and it can take minutes to deliberate; the Navigator and Science Operator manage the execution of sub-goals generated by the Mission Manager and could deliberate within a second. The temporal scope for both is in the order of a minute. The Executive encapsulates access to functional layer commands and state variables. It is approximated as having zero latency with no deliberation.

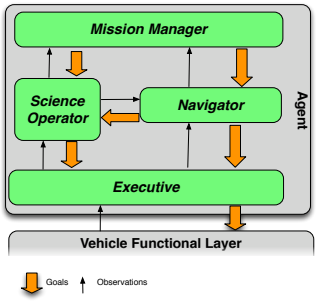

Fig. 3: An agent is composed of multiple reactors or control loops

4 Partitioned Control

Partitioning exploits a divide-and-conquer principle where each reactor manages a component of the agent control problem thereby allowing deliberation and reaction at different rates, over different time horizons and on different state variables. More importantly, partitioning *allows plan failures to be localized within a control loop without exposure to other parts of the system.* In this section we present the overall control loop for coordinating these reactors and detail how synchronization of state is accomplished efficiently.

4.1 The Agent Control Loop

The agent coordinates the execution of reactors within a global control loop based on the SPA paradigm. The *uses* and *owns* relations provide the necessary detail to direct the flow of information between reactors. Observations flow from each reactor that *owns* a state variable to all reactors that *uses* it during a process called *synchronization*. Goals flow from reactors that *uses* a state variable to the reactor that *owns* it through a process called *dispatching*. Dispatching involves posting tokens from each external timeline *e* in a complete plan as goals for the reactor *r* where *owns(r, s(e))* over a horizon given by h_r in (1). *Deliberation* is the process of planning an evolution of state variables from current values to requested values. Synchronization, deliberation and dispatching are steps of the core agent control loop described in Algorithm 1.

The agent executes the set of reactors, R, concurrently. Execution of the loop occurs once per tick. Each step begins at the start of a tick by dispatching goals from users to owners. The agent then synchronizes state across all reactors in $\mathcal{R}$ at the execution frontier, τ. If there is deliberation to be done, it must be preemptible by a clock transition which occurs as an exogenous event.

Algorithm 1 The agent control loop

RUN$(\mathcal{R}, \tau, \Pi)$

1 **if** $\tau \geq \Pi$ **then return** ; *// If the mission is over, quit*
// Dispatch goals for all reactors for this tick
2 DISPATCH$(\mathcal{R}, \tau)$;
// Synchronize. If no reactor left afterwards, quit
3 $\mathcal{R}' \leftarrow$ SYNCHRONIZEAGENT$(\mathcal{R}, \tau)$; *// Alg. 2*
4 **if** $\mathcal{R}' = \emptyset$ **then return** ;
// Deliberate in steps until done or the clock transitions
5 $\delta \leftarrow \tau + 1$;
6 $done \leftarrow \bot$;
7 **while** $\delta > \tau \wedge \neg\, done$
8 **do** $done \leftarrow$ DELIBERATE$(\mathcal{R}', \tau)$;
9 **while** $\delta > \tau$ **do** SLEEP; *// Idle till the clock transitions*
// Tail recursive, with possibly reduced reactor set
10 RUN$(\mathcal{R}', \tau, \Pi)$;

4.2 Synchronization

In a partitioned control structure, opportunities for inconsistency exist since each reactor has its own representation for the values of a state variable. While we allow divergent views of future values of a timeline to persist, we require that all timelines converge at the execution frontier after synchronization. For an agent to be complete, agent state variables must have a valid value at τ. It is the responsibility of the owner reactor to determine this value during synchronization and the responsibility of users of that state variable to reconcile their internal state with this observation. This explicit ownership specification enables conflict resolution.

Requirements

The concept of a flaw [Bedrax-Weiss *et al.*, 2003] i.e a potential inconsistency that must be resolved, is central to the

definition of synchronization. We are concerned with flaws that may render the state of the reactor inconsistent at the execution frontier. More specifically, we are concerned with any token *t* that *necessarily* impacts any timeline *L* of a reactor *r* at the execution frontier τ, which has not been *inserted* in $\mathcal{Q}(L)$. Formally a flaw is a tuple $f = (t, L)$ that is resolved by *insertion* in $\mathcal{Q}(L)$.

Definition 3 *A reactor r is synchronized for τ, denoted $\textcircled{S}(r, \tau)$, when the following conditions are satisfied:*

- *All flaws are resolved at τ when $\mathcal{F}_\tau(r) = \emptyset$, where $\mathcal{F}_\tau(r)$ returns an arbitrarily ordered set of flaws of a reactor for synchronization at the execution frontier. A formal definition of $\mathcal{F}_\tau(r)$ follows in Definition 4.*
- *All internal and external timelines have a unique valid value at τ, i.e there are no "holes" or conflicts in the timeline:*

$$\forall L \in (\mathcal{I}_r \cup \mathcal{E}_r), \exists o \in \mathcal{T}(L) : L(\tau) = \{o\} \quad (2)$$

- *All external timelines have the same value as the corresponding internal timeline of the owner reactor at τ. This ensure that every reactors share a consistent view of the current state of the world.*

Synchronization of the agent means synchronization of its reactors: $\forall r \in \mathcal{R} : \textcircled{S}(r, \tau)$

Assumptions

We introduce a number of reasonable assumptions to reduce synchronization complexity to polynomial time. Our approach builds on the semantics of the partitioned structure to enable synchronization of the agent via incremental local synchronization of each reactor.

From the perspective of a reactor, observations are taken as facts that are exogenous and monotonic: once an observation is published/received, it cannot be retracted. We call this the *Monotonicity Assumption (MA)*. It implies that a reactor r will publish an observation o starting at τ only if it *owns* the corresponding timeline and it has been fully synchronized for this tick ($\textcircled{S}(r, \tau) = \top$).

The last observation made on an *external* timeline is valid until a new observation is received. We call this the *Inertial Value Assumption (IVA)*. It implies that once a reactor has received all the observations for its external timelines at τ, all external timelines $E \in \mathcal{E}_r$ that do not have an associated observation o for this tick will consider that the previous state still holds :

$$(\nexists o \in \mathcal{O}_r \cap \mathcal{T}(E) : o.start = \tau) \Rightarrow E(\tau).end > \tau \quad (3)$$

This assumption ensures that all values of an external timeline contain an observation up till τ. Consequently we require an initial observation at $\tau = 0$. In the case of internal timelines, the model $\mathcal{M}_w$ must specify the value to assign in all cases. A corollary of the MA is that the IVA should not be applied for a reactor r_1 until:

$$\forall r_2 \in \mathcal{R}, r_1 \rhd r_2 : \textcircled{S}(r_2, \tau) \quad (4)$$

where $\rhd$ is a relation defined as follows:

$$\forall r_1, r_2 \in \mathcal{R} :$$
$$\big(\exists s \in \mathcal{S}; uses(r_1, s) \wedge owns(r_2, s)\big) \Rightarrow r_1 \rhd r_2$$
$$(\exists r_3 \in \mathcal{R}; r_1 \rhd r_3 \wedge r_3 \rhd r_2) \Rightarrow r_1 \rhd r_2$$

(4) imposes a strong relation between the synchronization of a reactor and the structure in the dependency graph (defined by $\rhd$). Indeed, even though we could allow cyclic dependencies between reactors this would require iteration though this graph until we find a fixed point where all reactors are synchronized.

To avoid thus we ensure $\rhd$ is a partial order (ie $r_1 \rhd r_2 \Rightarrow \neg(r_2 \rhd r_1)$) Consequently, all reactors are distributed across a directed acyclic graph (DAG) where the root nodes are the reactors of $R_{\mathcal{W}}$. This *Acyclic Dependency Assumption* (ADA) ensures synchronization is achievable via iteration through a linear transformation of the partial order $\rhd$ (denoted by $R[i]$). Where state variables are cyclically dependent, they must be owned by the same reactor. Absent this assumption, global synchronization requires iterative local synchronization to a fixed point which cannot guarantee polynomial time convergence.

Determining how to arrive at a suitable partition design is challenging; [Salido and Barber, 2006] offers a promising direction in using constraint cliques.

Synchronizing the Agent

Algorithm 2 shows how the agent is synchronized by synchronizing each reactor in the order defined by $R[i]$. It is possible that a reactor may fail to synchronize. Such a failure implies that no consistent and complete assignment of values was possible for its timelines, and usually indicates an error in the domain model. Under these conditions, the agent must remove the reactor as well as all its dependents from the control structure in order to satisfy the requirements for consistent and complete state. This failure mode offers the potential for graceful degradation in agent performance with the possibility of continued operation of reactors implementing safety behaviors.

Algorithm 2 Agent synchronization

```
SYNCHRONIZEAGENT(R, τ)
1  R_in ← ∅;
2  R_out ← ∅;
3  for i ← 1 to SIZE(R)
   // Get next reactor on dependency list
4        do r ← R[i];
5           if (∃r_out ∈ R_out : r ▷ r_out)
                ∨¬SYNCHRONIZE(r, τ) // Alg. 3
            // If r cannot be synchronized exclude it
6              then R_out ← R_out + r;
7              else R_in ← R_in + r;
   // Return the reactors that are still valid
8  return R_in;
```

Synchronizing a Reactor

Algorithm 3 describes synchronization of a reactor. Synchronization begins by applying IVA to extend the current values of external timelines with no new observations according to (3). *Relaxation* is required if planning has taken longer than permitted ($> \lambda_r$) or if there is no complete and consistent refinement of the set of flaws at the execution frontier (see Algorithm 4). Relaxation decouples restrictions imposed by

planning from entailments of the model and execution state by deleting the plan but retaining observations and committed values. Goals must be re-planned in a subsequent deliberation cycle. After relaxation, a second attempt is made to complete the execution frontier. If this fails, synchronization fails and the reactor will be taken off line by the agent. If this succeeds, the reactor will iterate over its internal timelines, publishing new values to its users. Finally, at every step of synchronization, a garbage collection algorithm is executed cleaning out tokens in the past with no impact to the present or future. Details of garbage collection and plan relaxation are outside the scope of this paper.

Algorithm 3 Single reactor synchronization

SYNCHRONIZE(r, τ)

// Apply IVA to extend current observations
for each E $\in \mathcal{E}_r$
  do if $(\nexists o \in \mathcal{O}_r \cap \mathcal{T}(E); o.start = \tau)$
    then $E(\tau).end \leftarrow E(\tau).end \cap [\tau + 1, \infty]$;
// Complete execution frontier
if $(\tau \in h_r \vee \neg$COMPLETE$(r, \tau))$ *// Alg. 4*
  then if $(\neg$RELAX$(r, \tau) \wedge \neg$COMPLETE$(r, \tau))$ *// Alg. 4*
    then return $\bot$
// Publish new state values to users of internal timelines
for each I $\in \mathcal{I}_r$
  do $State \leftarrow I(\tau)$;
    if $State[0].start = \tau$
      then for each E $\in \varepsilon(s(I))$
        do $\mathcal{T}(E) \leftarrow \mathcal{T}(E) \cup State$;
          $\mathcal{O}(E) \leftarrow \mathcal{O}(E) \cup State$;
// Clean out tokens in the past that have no impact
GARBAGECOLLECT(r, τ);
return $\top$;

Resolving Flaws at the Execution Frontier

We now define $\mathcal{F}_\tau$ (used in Definition 3) and describe its application in the function RESOLVEFLAWS used to synchronize a reactor. The components of this definition are:

- the temporal scope of the execution frontier which we define to include the current state (i.e. tokens that contain τ) and the prior state (i.e. tokens that contain $\tau - 1$).
- an operator $\mathcal{F}_\pi$ which returns the set of flaws for deliberation. Flaws in $\mathcal{F}_\pi$ should not be in $\mathcal{F}_\tau$. The intuition is to check if a token is a goal, or if it there is a path from the token to a goal in the causal structure of the plan.
- a unit decision operator $\mathcal{U}$ for a flaw, f, that excludes flaws that can be placed at more than one location around τ in $\mathcal{Q}(f.L)$:

$$\mathcal{U}(f) \Rightarrow \left(\begin{array}{c} \forall q_1, q_2 \in (f.L(\tau - 1) \cup f.L(\tau)) : \\ (q_1 \otimes f.t) \wedge (q_2 \otimes f.t) \Rightarrow q_1 = q_2 \end{array} \right)$$

where $\otimes$ indicates that two tokens can be *merged*:

$$p_1(s_1, e_1, \overrightarrow{x_1}) \otimes p_2(s_2, e_2, \overrightarrow{x_2}) \Rightarrow (p_1 = p_2) \wedge (s_1 \cap s_2 \neq \emptyset) \wedge (e_1 \cap e_2 \neq \emptyset) \wedge (\nexists x \in \overrightarrow{x_1} \cap \overrightarrow{x_2}; x = \emptyset)$$

Definition 4 *For a given reactor r, the set of synchronization flaws $\mathcal{F}_\tau(r)$ is defined by the set of flaws $f \notin \mathcal{F}_\pi(r)$ that overlaps the execution frontier τ and are unit decisions ($\mathcal{U}(f) = \top$).*

The call to the function RESOLVEFLAWS(r, τ) iteratively selects one of the flaws in $\mathcal{F}_\tau(r)$ and resolves it by insertion in its timeline until $\mathcal{F}_\tau(r)$ is empty. Insertion may be infeasible indicating that no complete and consistent refinement of the current execution frontier is possible. [Bernadini and Smith, 2007] describes token insertion in a partial plan.

Completion

Algorithm 4 utilizes RESOLVEFLAWS to complete synchronization. It begins by resolving all the available flaws. Resolution of the set of flaws is a necessary condition for completeness. However, it is not a sufficient condition for two reasons. First, it is possible that holes may exist in the internal timelines (application of IVA ensures that all external timelines are complete) that must be filled. Second, it is possible that the end time for the current token in an internal timeline I, is an interval. This must be restricted so that $I(\tau)$ returns a singleton after synchronization; see (2). To address this we define a policy to complete an internal timeline under these conditions. For the first case, MAKEDEFAULTVALUE will generate a default value according to the model which is inserted to fill the hole. In the second case, the end time of the current value will be extended. These modifications may generate more flaws in turn. For example, tokens previously excluded from synchronization by $\mathcal{U}$ may now become unit decisions. Furthermore, additional rules in the model may now apply. Consequently, we invoke RESOLVEFLAWS again.

Algorithm 4 Completion of the execution frontier

COMPLETE(r, τ)

if $\neg$RESOLVEFLAWS(r, τ)
  then return $\bot$
// Complete Internal Timelines
for each I $\in \mathcal{I}_r$
  do v $\leftarrow I(\tau)$;
    // Fill with default value if empty
    if v = $\emptyset$
      then INSERT(I, MAKEDEFAULTVALUE(I, μ_r, τ));
      else $v[0].end \leftarrow v[0].end \cap [\tau + 1, \infty]$;
return RESOLVEFLAWS(r, τ);

4.3 Complexity Analysis

We now consider the complexity of synchronizing an agent. The key result is that synchronization is a polynomial time multiple of the cost of primitive operations on a plan. We assume the following operators are executed in amortized constant time:

- $\mathcal{F}_\tau$ The operator to obtain the sequence of flaws in the execution frontier.
- INSERT(L, t) The procedure to insert a token t in timeline L.
- MAKEDEFAULTVALUE(L, μ_r, τ) The procedure to generate a default token for timeline L.

We further assume:

- GARBAGECOLLECT is linear in the number of tokens in the past that have not yet been removed.
- RELAX is linear in the number of tokens in all timelines.
- Insertion of a token by merging with an existing token generates no new flaws.
- The costs of $\otimes, \cap, \cup$ as used in synchronization are bounded and negligible.

Consider the procedure RESOLVEFLAWS. This procedure is linear in the number of flaws, since for each flaw encountered, it is resolved by an insertion operation within amortized constant time with no backtracking. Assume a reactor *r* has $\mathcal{N}_r$ timelines. In the worst case, every timeline in a reactor will require a new value for the current and prior tick. Assume that in the worst case, every new value generates a flaw for every other possible position in all timelines in the execution frontier (i.e. $2\mathcal{N}_r$ -1 flaws per new value). This gives a maximum complexity for RESOLVEFLAWS of $2\mathcal{N}_r \times (2\mathcal{N}_r$-1) or O($\mathcal{N}_r{}^2$). In the worst case, the procedure COMPLETE calls RESOLVEFLAWS twice. However, if simply refining the execution frontier, this does not change the cumulative number of flaws. Since iteration over the internal timelines is linear in a value $\leq \mathcal{N}_r$, we have a complexity of O($\mathcal{N}_r{}^2$) for Algorithm 4.

In the worst case, synchronization of a reactor (Algorithm 3) incurs the following costs:

- O($\mathcal{N}_r$) to complete external timelines
- O($\mathcal{N}_r{}^2$) to call COMPLETE the first time, which we assume will fail.
- O(P_r) to RELAX the plan where P is the number of tokens in the plan.
- O($\mathcal{N}_r{}^2$) to call COMPLETE the second time, which we assume will succeed.
- O($\mathcal{N}_r$) to publish observations
- O(H_r) to garbage collect where H is the number of tokens in the plan that have passed into history.

Since synchronization of the agent is accomplished by iteration over the set of reactors, without cycling, the worst case time complexity for synchronization is given by:

$$O(\text{Ⓢ}) = O(\sum_{r \in \mathcal{R}} \mathcal{N}_r^2 + P_r + H_r) \quad (5)$$

5 State Estimation

we have embedded a Hidden Markov Model (HMM) [Rabiner, 1986] for environmental state estimation that seamlessly drives online adaptation of the agent [McGann *et al.*, 2008a]. This HMM is encoded within the unified representational and computational framework of a Deliberative Reactor. The sensor data depends on the feature of interest; e.g., we use three optical sensor readings in conjunction with altitude and depth of the vehicle for feature detection. HMMs are useful since the stochastic nature of these models can correlate the type of features we want to detect with the sensor observations. Further, such models can also be learned using past mission data [Fox *et al.*, 2007].

One methodology to abstract features of interest from large data sets is *clustering*. There are several clustering techniques requiring different levels of expert supervision. We use a semi-supervised learning method [Zhu, 2005] to extract an environmental model that make uses of both labeled and unlabeled data for training. Compared to unsupervised techniques, the learning algorithm exploits labeled information reflecting greater prior knowledge and requires less effort compared to supervised methods to categorize data by extrapolating labeled data over large data sets.

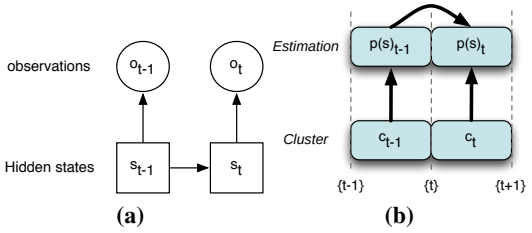

Fig. 4: Similarity between HMM and Timeline Representations

Our clustering algorithm allows us to extract a set of clusters with an associated probability $p(c \mid s)$ to observe cluster c knowing the current label (e.g. either inside or outside the feature of interest). This result can then be used to learn an HMM [Fox *et al.*, 2006]. Such a model exploits the dynamics of the observations to improve state estimation. Such feature detection computes the set of probabilities $\{p(in)_t, p(out)_t\}$ based on Markovian assumptions:

$$p(s)_t = a_t * p(c_t \mid s) * \sum_{k \in \{in,out\}} p(s \mid k) * p(k)_{t-1} \quad (6)$$

with a normalization factor a_t such that $\sum_{k \in \{in,out\}} p(s)_t = 1$.

The above equations encapsulate HMM based belief. In so doing, they express a constraint between current estimated state, the last observation and the previous estimated state. Fig. 4a shows an HMM execution trace representation where arrows represent the dependence between states and observations. Fig. 4b shows the timelines managing our state estimation where arrows stand for constraint propagation between tokens, indicating a strong similarity between the two.

The model is applied in execution as follows:

1. Sensor data is classified by the VCS so that it computes the corresponding cluster as in [Fox *et al.*, 2007]. This cluster value is provided as an observation on the Cluster timeline to the Navigator.
2. The observation is inserted in the plan during synchronization.
3. The implications of this observation on the Estimation timeline, expressed in the model, are applied and resolved during synchronization. Constraints then propagate the probability distribution according to (1). At this stage we have the estimation deduced from the HMM. The transition from the estimation to decision making can then be done directly. For instance to decide when to take a water sample the steps are:
4. If the probability of being within a feature of interest (such as an INL) dominates the distribution and other sampling conditions are satisfied (e.g spatial constraints dispersing sampling locations), then a sample should be taken. This rule is evaluated during synchronization as the constraint network is propagated.

5. Should this rule fire, a new token will be generated to transition the SamplerController into a firing state. This triggers a new deliberation phase to resolve the implications of this transition (i.e., select a sampling canister, and generate an action to fire it).

We use a similar process for altering the spatial resolution of the survey using the HMM to compute the probability of having seen the feature during the last vehicle transect. This in turn is used to determine the spatial separation between the previous and next transect.

6 Experimental Results

In this section we evaluate the performance of synchronization in partitioned and non-partitioned control structures. When plan operations are constant time, we show that synchronization is $O(N^2)$ in the worst case. Moreover, we demonstrate that actual costs of synchronization are accrued based on what changes at the execution frontier, making it efficient in practice. We further demonstrate that since operations on a plan are typically not constant time, but vary in diverse and implementation dependent ways according to plan size, partitioning control loops can substantially reduce the net cost of synchronization and deliberation.

The following results are based on our current implementation of the framework with each reactor using the same CTP based planner using chronological-backtracking refinement search to deliberate. The agent invokes each reactor as defined in the Algorithm 1. We use a single model shared by all reactors. It contains a timeline type with a token type given by the predicate $p(start, end)$ with no extra parameters. Our experiments were executed on a MacBook Pro running at 2 Ghz.

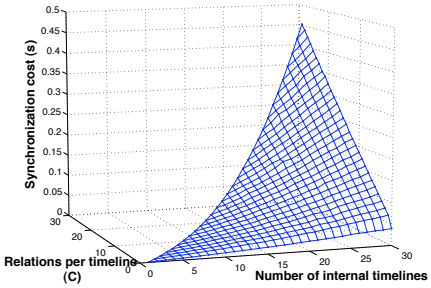

Fig. 5: The relationship between synchronization time, number of constraints and number of internal timelines.

Fig. 5 shows results of a run with a single reactor configuration where each problem instance runs for 50 ticks. A set of problem instances were generated by modifying the number of internal timelines (I) and the connectivity of the constraint graph (C). There is no deliberation involved. CPU usage for synchronizing the agent is measured at every tick, and averaged over all ticks. The figure shows that average synchronization cost increases linearly in I and C and quadratically as the product of I and C. Further Fig. 6 clearly shows that synchronization costs vary linearly as a function of the size of the overlap of external timelines.

Fig. 7 shows the impact of partitioning on problem solving, by varying the partitioned structure of an agent without changing the number of timelines being synchronized. We use 120 internal timelines spread evenly across all reactors. For each problem instance we distributed the timelines between a varying number of reactors between 1 and 120. With 120 reactors there is only 1 timeline per reactor and the agent control structure is maximally partitioned. Deliberation fills out the timeline with 1 token per tick for 10 ticks with no search required. For each problem, the cumulative synchronization and deliberation CPU usage was measured. The number of state variables remains the same as timelines are apportioned between reactors showing the impact on problem solving by partitioning.

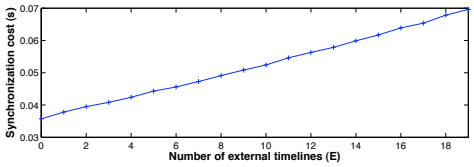

Fig. 6: Information sharing via external timelines results in increased cost of synchronization linear in the number of external timelines.

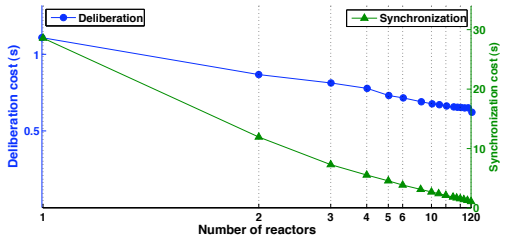

Fig. 7: Synchronization costs associated with partitioning.

Partitioning allows us to model the system so as to localize changes within a reactor to cap costs associated with synchronization. Furthermore, timelines at higher levels of abstraction necessarily change more slowly relative to timelines at lower levels of abstraction making partitioning more effective. To explore this, we define a system with 4 reactors, each having 10 internal timelines, and all but the bottom reactor having 1 external timeline. The token duration of internal timelines is given by 2^d where d is the depth of the reactor in its hierarchy (maximum depth was 3). Fig. 8 illustrates fluctuating synchronization costs with variation in the rate of change of timelines in the agent with the highest cost being when all timeline changes occur together.

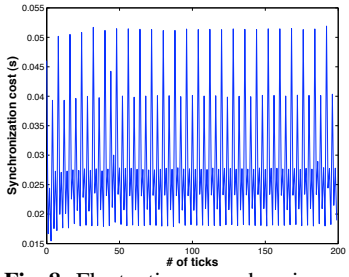

Fig. 8: Fluctuating synchronization costs across time

Our framework has been integrated onboard an Autonomous Underwater Vehicle and deployed for science exploration missions at sea in Monterey Bay, California. A number of field results described in [McGann *et al.*, 2008b] and at [TREX, 2008] bear out the scalability of the approach in practice and demonstrate the promise of partitioned controllers in real-world dynamic environments. T-REX included 28 timelines distributed across 3 reactors. In all missions to date, the deliberative reactors run on a 367 MHz EPX-GX500 AMD Geode stack using Red Hat Linux, with the VCS functional layer running on a separate processor.

In November 2008 on a cruise over the Monterey Canyon, our objective was to demonstrate that T-REX was able to carry out a nominal science mission and be accurate in its es-

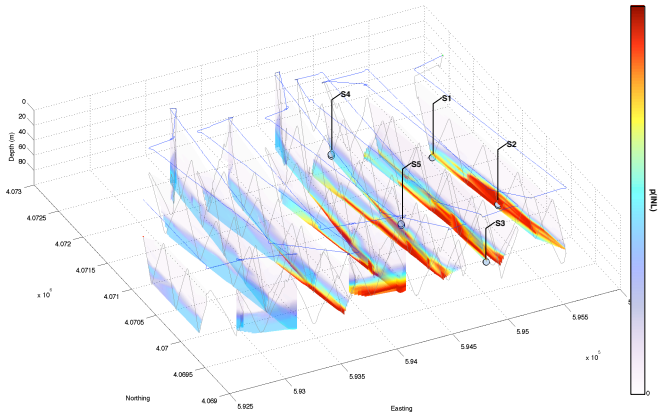

Fig. 9: Visualization of a T-REX run over the Monterey Canyon in November 2008. Red indicates high probability of INL presence as detected by onboard sensors. S1-S5 indicate triggering of 10 water samplers two at a time.

timation of an Intermediate Nephaloid Layer (INL). During the uninterrupted 6 hours and 40 minutes run at sea, T-REX was able to identify the INL and bring back water samples. Fig. 9 shows the vehicles transect, the context of presence of the INL in the water-column detected by onboard sensors. The figure also shows the vehicle's change in sampling resolution, starting with high resolution transects and ending with a low resolution survey (where the INL is not visible), and the water samples taken two at a time within the feature of interest.

7 Conclusion

We introduce a formal framework for specifying an agent control structure as a collection of coordinated control loops while decoupling deliberation over goals from synchronization of agent state. We present algorithms for integrated agent control within this partitioned structure showing how HMM based approaches can impact planning. For weakly coupled reactors partitioning can offer substantial performance improvements with the overhead of information sharing. Further partitioning allows plan failures to be localized within a control loop without exposure to other parts of the system, while localized reactor failure can ensure graceful system degradation.

8 Acknowledgments

This research was supported by the David and Lucile Packard Foundation. We are grateful to MBARI oceanographer John Ryan in helping formulate and drive our technology and our engineering and operations colleagues for their help in integration and deployment. We thank NASA Ames Research Center for making the EUROPA, Willow Garage for supporting McGann's work on T-REX and the Spanish government for Olaya's fellowship at MBARI.

References

[Alami *et al.*, 1998] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An Architecture for Autonomy. *The International Journal of Robotics Research*, Jan 1998.

[Ambros-Ingerson and Steel, 1988] J. Ambros-Ingerson and S. Steel. Integrating Planning, Execution and Monitoring. *Proc. 7th AAAI*, Jan 1988.

[Bedrax-Weiss *et al.*, 2003] T. Bedrax-Weiss, J. Frank, A.K. Jonsson, and C. McGann. Identifying Executable Plans. In *Workshop on Plan Execution, ICAPS*, 2003.

[Bernadini and Smith, 2007] S. Bernadini and D. Smith. Developing Domain-Independent Search Control for Europa2. In *Proc. ICAPS-07 Workshop on Heuristics for Domain-independent Planning*, 2007.

[Chien *et al.*, 1999] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Integrated Planning and Execution for Autonomous Spacecraft. *IEEE Aerospace Conference*, 1:263–271 vol.1, 1999.

[Finzi *et al.*, 2004] A. Finzi, F. Ingrand, and N. Muscettola. Model-based Executive Control through Reactive Planning for Autonomous Rovers. In *Proc. IROS*, 2004.

[Fox *et al.*, 2006] M. Fox, M. Ghallab, G. Infantes, and D. Long. Robot introspection through learned hidden markov model. *Aritificial Intelligence Journal*, 170(2):59113, 2006.

[Fox *et al.*, 2007] M. Fox, D. Long, F. Py, K. Rajan, and J. P. Ryan. In Situ Analysis for Intelligent Control. In *Proc. of IEEE/OES OCEANS Conference*. IEEE, 2007.

[Frank and Jónsson, 2003] J. Frank and A. Jónsson. Constraint-based Attribute and Interval Planning. *Constraints*, 8(4):339–364, oct 2003.

[Gat, 1998] E. Gat. On Three-Layer Architectures. In D. Kortenkamp, R. Bonnasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*, pages 195–210. MIT Press, 1998.

[Haigh and Veloso, 1998] K. Haigh and M. Veloso. Interleaving Planning and Robot Execution for Asynchronous User Requests. *Autonomous Robots*, Jan 1998.

[Knight *et al.*, 2001] R Knight, F Fisher, T Estlin, and B Engelhardt. Balancing Deliberation and Reaction, Planning and Execution for Space Robotic Applications. *Proceedings of Intelligent Robotics and Systems*, Jan 2001.

[McGann *et al.*, 2008a] C. McGann, F. Py, K. Rajan, J. P. Ryan, and R. Henthorn. Adaptive Control for Autonomous Underwater Vehicles. In *AAAI*, Chicago, IL, 2008.

[McGann *et al.*, 2008b] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. McEwen. Preliminary Results for Model-Based Adaptive Control of an Autonomous Underwater Vehicle. In *Int. Symp. on Experimental Robotics*, Athens, Greece, 2008.

[Muscettola *et al.*, 1998] N. Muscettola, P. Nayak, B. Pell, and B. Williams. Remote Agent: To Boldly Go Where No AI System Has Gone Before. In *AI Journal*, volume 103, pages 5–48, 1998.

[Muscettola *et al.*, 2002] N. Muscettola, G. Dorais, C. Fry, R. Levinson, and C. Plaunt. IDEA: Planning at the Core of Autonomous Reactive Agents. In *IWPSS*, October 2002.

[Rabiner, 1986] L. R. Rabiner. An introduction to hidden markov models. *IEEE ASSP Magazine*, pages 4–16, 1986.

[Rajan *et al.*, 2000] K. Rajan, D. Bernard, G. Dorais, E. Gamble, B. Kanefsky, J. Kurien, W. Millar, N. Muscettola, P.P. Nayak, N. Rouquette, B. Smith, W. Taylor, and Y. Tung. Remote Agent: An Autonomous Control System for the New Millennium. In *Proc. Prestigious Applications of Intelligent Systems, ECAI*, 2000.

[Ryan *et al.*, 2005] J. P. Ryan, F. P. Chavez, and J. G.

Bellingham. Physical-biological coupling in monterey bay, california: Topographic influences on phytoplankton ecology. *Marine Ecology Progress Series*, 287:23–32, 2005.

[Salido and Barber, 2006] M. A. Salido and F. Barber. Distributed CSPs by Graph Partitioning. *Applied Mathematics and Computation*, 183:212–237, 2006.

[Thomas, 2006] *et al* Thomas, H. Mapping auv survey of axial seamount. *Eos Trans. AGU*, 87(52)(Fall Meet. Suppl., Abstract V23B-0615), 2006.

[TREX, 2008] TREX. http://www.mbari.org/autonomy/trex/. 2008.

[Yuh, 2000] J. Yuh. Design and Control of Autonomous Underwater Robots: A Survey. *Autonomous Robots*, 8:7–24, 2000.

[Zhu, 2005] X. Zhu. Semi-supervised Learning Literature Survey,Technical Report 1530, Dept. of Computer Sciences, University of Wisconsin-Madison. Technical report, 2005.