Model-based, Hierarchical Control of a Mobile Manipulation Platform

Conor McGann, Eric Berger, Jonathan Bohren, Sachin Chitta, Brian Gerkey, Stuart Glaser, Bhaskara Marthi, Wim Meeussen, Tony Pratkanis, Eitan Marder-Eppstein, Melonee Wise
Willow Garage (mcgann@willowgarage.com)

Abstract

PR2 is a sophisticated mobile manipulation platform designed for operation in dynamic and unstructured indoor environments. In this paper we describe an experiment using TREX, a hierarchical control framework based on constraint-based temporal planning, to coordinate PR2 behavior. The experiment was part of a fully integrated demonstration of PR2 capabilities involving autonomous navigation, door-opening, and recharging using standard electrical outlets. The goal of this experiment was to evaluate the applicability of a model-based, planning centric approach for practical robotics on a large scale. The results were encouraging. Not only did TREX play an important role in accomplishing the milestone, which was in itself a significant achievement in autonomous robotics, but it did so with modest computatational overhead and system complexity. In this paper we outline the details of the milestone and how TREX was used to achieve it, providing a quantitative and qualititative evaluation of TREX performance. We believe this presents a promising pathway for deep integration of declarative models, and automated planning as a paradigm for practical robot programming. All software described in this paper, including TREX, is available under an Open Source license, via `http://ros.sourceforge.net`.

Introduction

PR2 is a sophisticated mobile manipulation platform designed for operation in dynamic and unstructured indoor environments. Writing programs to enable such a robot to operate autonomously, competently and robustly is hard. The overall goal of our work is to make programming such robots easier. Our strategy for achieving this goal is to develop robust, re-usable action primitives that are maximally decoupled, and focus on methods to compose these actions in a principled way to accomplish higher level tasks.

In this paper we report results from an experiment applying TREX ((McGann et al. 2008b), (McGann et al. 2008a)), a hierarchical control framework based on constraint-based temporal planning, to coordinate PR2 behavior. TREX was chosen because it claimed to support a synthesis of goal-directed and reactive behavior in a uniform computational model based on formal, declarative models and automated planning techniques. Moreover, EUROPA (Frank and Jónsson 2003), the constraint-based temporal planning system at the heart of TREX, has a promising track record in planning for practical robotics applications (Muscettola et al. 1998), (Bresina et al. 2005).

The experiment was part of a fully integrated demonstration of PR2 capabilities involving autonomous navigation, door-opening, and recharging using standard electrical outlets. The goal of this experiment was to evaluate the applicability of a model-based, planning centric approach for practical robot programming on a large scale.

Figure 1: PR2 plugging in to a standard outlet.

The paper is structured as follows. We begin with a description of PR2 and the demonstration requirements that motivated our work. Next we review alternative approaches for building an *Executive* for PR2 and indicate why TREX was chosen. We then describe TREX and present selected details of its application to this domain to convey how it worked in practice. This is followed by a presentation of quantitative and qualitative results which are the main contribution of the paper. We close with a discussion of our results and their implications for future work.

Requirements

Our experimental system is an alpha prototype of the PR2 mobile manipulation platform (Fig. 1). The PR2 comprises an omni-directional wheeled base, telescoping spine, two force-controlled 7-DOF arms and an actuated sensor head. Each arm has a 1-DOF gripper attached to it. The robot can negotiate ADA-compliant[1] wheelchair-accessible environments, and its manipulation workspace is similar to that of an average-height adult.

[1] `http://www.ada.gov/`

The sensor head comprises a Hokuyo UTM-30 planar laser range-finder on a tilt stage, and a pan-tilt stage holding both a Videre stereo camera and a 5 mp Prosilica camera. The laser is tilted up and down continuously, providing a 3D view of the area in front of the robot. The resulting point clouds are input to our perception system, which in turns drives our manipulation system. A second Hokuyo UTM-30 laser sensor, attached to the base, is used for navigation.

The PR2 carries multiple computers, connected by a gigabit LAN. The current computing configuration is four dual-core 2.6GHz machines running Linux. One of the four machines is modified to run a real-time kernel, providing a guaranteed 1KHz control loop, via EtherCAT,[2] over the robot's motors and encoders. PR2 uses ROS (Robot Operating System) (Quigley et al. 2009) for synchronous and asynchronous inter-process communication.

The Challenge

This experiment was conducted in the context of a challenge designed to test the capabilities of PR2 and ROS, and show that the system could operate robustly and autonomously in an indoor office environment. This challenge consisted of two main parts. The first was a navigation marathon, conducted on a stripped down version of PR2 (no arms). This was largely a matter of making our core navigation behavior robust, with only limited demands for top-level control. The second part of the challenge, and the focus of this work, was more of a triathlon, integrating planar navigation, navigation through closed, partially-open, or fully open doorways, and recharging by plugging in to one of a range of standard electrical outlets located around our building. Given an occupancy grid map of the building, plus approximate locations of doors and outlets, PR2 had to autonomously navigate to each of 10 outlets, selected by a user at the start, whereupon it would plug itself in, unplug and move on to the next one. In the course of driving from one outlet to another, PR2 would traverse doorways as needed. If an outlet was in an office with a locked door, it could give up on that goal (or try again later). It had to accomplish this in under 2 hours. Once underway, human intervention was prohibited.

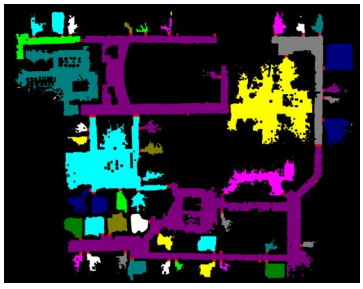


Figure 2: Regions in the topological map. Doorways are special regions, indicated in red. The map is derived from a metric map constructed from laser data.

[2] http://www.ethercat.org/

The Work Breakdown

The system was divided into 4 key sub-domains:

- Navigation. Handled planar navigation on a metric costmap derived from laser data. This provided a capability for global navigation around the building, and local positioning of the robot over smaller scales, each based on achieving a target 3-DOF pose.
- Doors. Handled doorway traversal, including door and handle detection, local navigation for positioning the base, and arm commands for contacting and manipulating the door and handle. These capabilities were encapsulated as a set of 10 durative actions.
- Plugs. Handled outlet and plug detection, local navigation around the outlet, as well as arm commands for grasping and manipulating the plug. The plug was mounted on the base via a magnet and would be removed and restored to that position on each recharge cycle. These capabilities were also encapsulated as a set of durative actions.
- Topological map. A topological map (Fig. 2) was developed providing a graph based representation of regions and connectors derived from the underlying metric map. This enabled path planning for higher-level control. The topological map was annotated with prior information about doors, outlets and reachable approach points for each.

Substantial capabilities were required in each area, involving innovations in perception, planning and control. However, their specifics are outside the scope of this paper. Rather, we consider the products of each sub-domain as modular building blocks for higher level control. This approach allowed us to develop and test specific capabilities, in a decoupled fashion, as well as script specific combinations within each domain, for testing and demonstration.

The role of an *Executive*

Explicit scripting of actions is suitable for the testing of specific action sequences. A more sophisticated integration is required, however, to achieve a greater degree of robustness. The system should be able to assemble all capabilities in a coherent manner and recover in the event of failure. Such behavior co-ordination and system configuration management at this level are the job for the *Executive*. Co-ordination must be driven by top-level goals (e.g. recharge at outlet 6). These goals must be planned in a prudent order to provide for efficient execution. Safety constraints must be obeyed. For example, to avoid collision when driving around, the tilt laser must be running and the arms must be stowed. Also, shared resources must be managed. For example specific configurations of realtime controllers are required for each action, e.g. untucking the arms (useful to obtain a clear view of the base when looking for the plug) uses a joint-space trajectory controller, whereas grasping the handle uses an effort controller.

It is almost possible to operate with purely sequential action execution. However, some actions *may* be executable in parallel (e.g. it is possible to switch controllers and change the configuration of the tilt laser at the same time), and some

actions *must* be executed in parallel (e.g. drive the base through the doorway while pushing the door open with the arm). Moreover, we foresee that future scenarios involving both arms will require a greater degree of co-ordination between concurrent actions than those mentioned above.

In summary, the *Executive* must be commanded by high-level goals, and integrate high-level planning, durative and concurrent actions, discrete and continuous states, substantial uncertainty in the duration that actions might take, and in their outcomes. In addition, actions are subject to timing, safety and resource constraints.

Related Work

There are many approaches to consider for building an *Executive*. We will consider purely reactive methods, three-layer architectures, and model-based, planning centric approaches.

Reactive, procedural methods

Pre-sequenced linear plans are out of the question due to the high-degree of uncertainty in action outcomes. For the most part, individual subdomains (i.e. plugs, doors and navigation) might be handled using purely reactive techniques encoded as finite-state machines or teleo-reactive programs (Nilsson 1994). However, these approaches require encoding all conditional behavior as part of the state machine, or part of the condition logic of a TR program, explicitly checking if the arm is tucked prior to driving (for example), and inserting the action to tuck the arms if it is not. Similarly, these procedural methods would have to check the controller configurations for mechanisms they require (i.e. base, head, arm) and generate suitable configuration switching statements as needed. Purely reactive methods do not allow for scheduling action execution to account for these requirements since they are only discovered as they are needed. Moreover, there is still the need for planning for higher level goals which makes integration of deliberation and reaction inevitable.

Three Layer Architectures

The most common method for integrating deliberative and reactive behavior is based on layered architectures that integrate deliberative planning at the top layer, reactive controllers at the bottom, and a sequencer in the middle to coordinate controller execution, guided by the plan. Examples of this approach abound (Gat et al. 1997). One limitation of 3-tier architectures is that they confine the role of formal, deliberative techniques to only the highest levels of abstraction and render subsequent robot behavior subject to informal, reactive approaches that do not consider problematic implications of local decisions. For our purposes, this would offer little advantage over reactive, procedural approaches. Furthermore, the separation of deliberative and reactive behaviors into separate layers with separate technologies and separate specifications compounds the complexity of integration.

Model-based, planning centric methods

More recently, a number of researchers have developed executives that emphasize model-based approaches and deep integration of automated planning (Shanahan 2000), (Beetz 2000), (Williams et al. 2003). IDEA (Muscettola et al. 2002) is notable for the richness of the underlying timeline-based semantics which are founded on constraint-based temporal planning (CTP) (Frank and Jónsson 2003), and the extent to which it exploits automated planning and plan-based reasoning techniques at the core of execution. CTP has been applied in a number of practical applications ((Muscettola et al. 1998), (Bresina et al. 2005)). IDEA was implemented on top of the open-source EUROPA [3] planning framework , and was fielded in a number of technology demonstrations (Aschwanden et al. 2006). EUROPA supports metric time and resources, durative and concurrent actions, discrete and continuous states, and an expressive and extendable language for temporal and non-temporal constraints.

TREX ((McGann et al. 2008b), (McGann et al. 2008a)) was developed initially for deployment on Autonomous Underwater Vehicles (AUVs) based on the same foundations as IDEA (i.e. the CTP paradigm and a deep integration of planning techniques in execution). TREX is distinct from IDEA in a number of ways. Most notably, it identifies scalability as a key issue and uses a formal framework for partitioning an agent structure into a collection of co-ordinated control loops to allow planning based approaches to scale to more complex applications efficiently. Secondarily, it utilizes a more direct mapping to the underlying EUROPA planning system which exposes greater functionality with less integration complexity. This is an important practical concern. Even though TREX had been successfully deployed on AUVs, it was an open question how well it would apply to a more complex and reactive robotic system like PR2 since it possesses more demanding requirements for reaction times than did previously deployed systems. After a brief review of the core concepts, we will describe how TREX was applied in practice to this domain.

Key TREX Concepts

TREX models the world as a set of state variables, whose evolution is captured in *timelines*. The work of planning and control is partitioned into a set of *reactors*, each of which implements a sense-plan-act-loop. Information flows between reactors via timelines. There are three main ideas to understand. First, the notion of how a timeline-based representation maps to execution. Second, how planning and planning techniques are deeply integrated in a control loop based on the Sense-Plan-Act model. And finally, how an agent control structure, founded on the semantics of timelines, can be partitioned and composed.

Timelines, Tokens and Timeline-based Execution

Informally, a timeline captures the trajectory of a state variable. The value of a state variable is described by a predicate and its temporal extent is captured by *start* and *end*

[3] http://babelfish.arc.nasa.gov/trac/europa

time-points. Time is broken up into discrete *ticks*. Listing 1 provides a NDDL [4] declaration for a sub-type of an *AgentTimeline* capturing the pose of a robot with position (x, y, z) and orientation (qx, qy, qz, qw)[5]. Instances of a predicate are referred to as *tokens*, which include variables for each predicate parameter as well as implicit variables for its temporal scope.

```
1 class Pose extends AgentTimeline {
2   predicate Holds{ float x, y, z, qx, qy, qz, qw; }
3 };
```

Listing 1: NDDL class declaration for Pose.

An external action can be modeled as a timeline with states for when it is *Active* or *Inactive*. For example, consider an action to move the robot to a target pose using planar navigation. The *Active* state indicates the action is actively pursuing a goal pose. The *Inactive* state indicates it is not active, and provides feedback, including if the action succeeded or failed in some way. Listing 2 illustrates how this is declared in NDDL. Note that predicate declarations allow values on timelines to have different structure if necessary. A model may also impose constraints on the evolution of state in a timeline. In this example, the temporal relations *meets* and *met_by* (Allen 1983) ensure the timeline cannot be switched directly from one active state to another without first becoming inactive.

```
1  class MoveBase extends AgentTimeline{
2    predicate Active{
3      float x, y, z, qx, qy, qz, qw;
4    }
5    predicate Inactive{
6      ResultStatus status;
7      float x, y, z, qx, qy, qz, qw;
8    }
9  };
10 MoveBase::Active{
11   met_by(Inactive);
12   meets(Inactive);
13 }
```

Listing 2: NDDL declarations for MoveBase action

A planner (perhaps a very trivial one) can be used to fill up timelines in the future. Fig. 3a) illustrates a plan containing 2 instances of a *MoveBase* action. Time-points are captured as intervals, reflecting flexibility or uncertainty about the duration of each action. The timeline is the data store for both planning and execution. The red arrow indicates the current tick in execution, called the *execution frontier*. As the clock advances, time bounds must be propagated approriately.

SPA and the Deliberative Reactor

Fig. 4 depicts an implementation of a Sense-Plan-Act control loop using constraint-based temporal planning. TREX refers to this as a *Deliberative Reactor* which is a special case of the more general idea of a *teleo reactor*. The *PlanDatabase* stores timelines and tokens for both planning and execution. It includes algorithms and data structures for consistency checking, and propositional inference to apply the model. The *Synchronizer* integrates observations with the plan. The *Planner* monitors the *PlanDatabase* for *flaws* and resolves them. *Flaws* arise due to inbound goals, and by implication of the model. The *Dispatcher* dispatches tokens in the plan as goal requests. The *Planner* and *PlanDatabase* are EUROPA components.

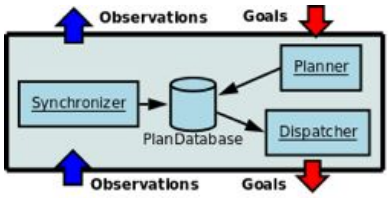


Figure 4: Internal structure of a Deliberative Reactor

Partitioning and Composition

The symmetry of inbound and outbound goals and observations enables natural composition of *reactors*. TREX defines an ownership and usage model of timelines to make the composition explicit, and the rules for information flow and conflict resolution unambiguous. If a reactor *owns* a timeline, it is solely responsible for deciding what value that timeline has as execution unfolds. Such timelines are *internal* to that reactor. If a reactor *uses* a timeline, it will receive new values for that timeline as they arise, and it may dispatch any goals it has for that timeline to the owner reactor. Such a timeline is *external* to its user. The user of a timeline *depends* on it's owner. This dependency dictates the flow of information in TREX. The key to scalability is that the scope of computation for each reactor is restricted to only the set of timelines it explicitly owns or uses, over the time horizon it cares to deliberate. This modularity, coupled with the dependency directed information flow, makes TREX amenable to divide-and-conquer strategies to scale up to larger scale systems efficiently in a unified computational framework.

Executive Design

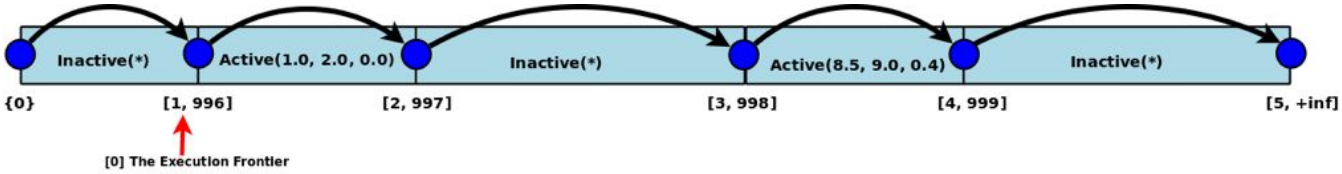
We now elaborate on the application of TREX to a challenging mobile manipulation demonstration. We first outline the overall structure, and then explore salient details from the domain to convey the richness and scale of the problem.

Reactor Graph

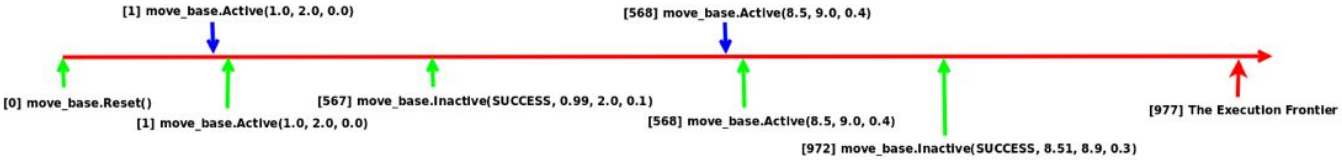
Fig. 5 indicates the reactor graph used for our PR2 experiments. It depicts the set of reactors and dependencies between them. Each reactor and its dependent links are color-coded. Goals flow in the direction of these links. Observations flow in the reverse direction. Each reactor is annotated with a name, its *look-ahead* indicating how far ahead to plan, its *latency* giving an upper bound on plan completion time,

[4] NDDL (pronounced 'noodle') is the modeling language of EUROPA and is directly used by TREX.

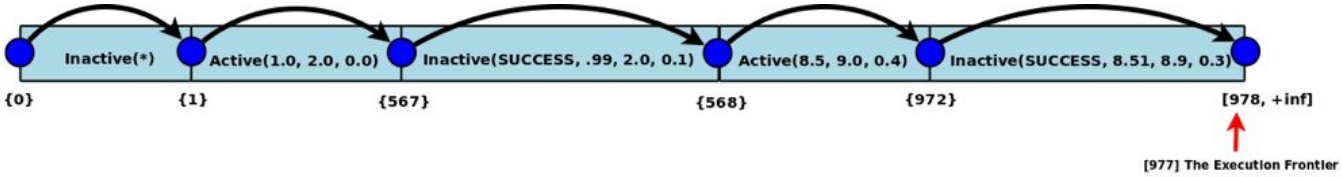
[5] Orientations are 4-tuples as we represent them as quaternions



(a) The move_base timeline after 2 goals have been planned. The timeline is a sequence of tokens (rectangles). A token is a temporally scoped predicate (the text in each rectangle). The temporal scope is captured with timepoints (indicated with circles). There is a precedence constraint (directed arc) between token timepoints. Each token has a minimum duration of 1 tick. Note that the timepoints are flexible. The latest end time for a goal is always less than the overall mission horizon for the agent (e.g. 1000). Each goal expresses that an external action is actively going to a target pose. The parameters for goals are bound. The expected feedback states (e.g. Inactive) do not have their parameters bound yet, though they can be constrained to precise values if desired. They will be bound in execution.



(b) Communication events between the external move_base action and the agent. The red line represents the advance of time from left to right. Perpendicular arrows are communication events occurring during execution. Blue arrows capture goal dispatch events. Green arrows capture observation notifications. The events are totally ordered, but are not to scale. At initialization, the move_base action's current state is published to the executive, indicating it is inactive. At tick 1, the first token in the plan is dispatched. This is manifest as a message to activate the move_base action where the parameters of the token become the goal arguments to move_base. Once activated, the move_base action will generate a message indicating it has transitioned to the state of active with the given goal parameters. This becomes an observation at tick 1 which is inserted in the plan by merging it with the existing planned token, and thus restricting the start time bound to a singleton. At tick 567, the move_base action achieved the goal pose within a given tolerance and thus transitioned inactive. This transition is published to the executive as an observation, which is inserted in the plan by merging the token. Now, unbound parameters in the plan are bound via intersection with the observed values from execution. At tick 568, the second goal is dispatchable. The same process repeats. In this example, there were no failures in execution.



(c) The resulting timeline at tick 977. Note that all timepoints and token parameters have been grounded. In practice, parts of the timeline that are in the past, and no longer impacting future tokens, can be discarded.

Figure 3: A timeline and its execution.

and parameters *i* and *e* indicating the number of internal and external timelines respectively.

The *Robot Control Subsystem* has no external timelines. It is mapped to a single exogenous state variable giving the planar pose of the robot, as well as each of 25 external action primitives previously described. This reactor is implemented as an adapter mapping ROS messages to goal requests, recalls and observations. All other reactors were instances of a *DeliberativeReactor* varying in their functional and temporal scopes. Real-time controller configuration management was handled in the *Mechanism Control* reactor. The *Doorman* encapsulated behavior for navigating doorways in the topological map. The *Driver* was used for navigation in all other regions (i.e. offices, hallways and open areas). The *Recharger* encapsulated all behavior for plugging in and unplugging. At the top level, the *Master* was used to plan the overall tour given high-level goals, and decompose these goals into successive calls to the *Doorman*, *Driver* and *Recharger*. The topological map played an important role, providing cost estimates for point-to-point navigation for the planner, and integrating data about doors and outlets to drive navigation. The *Safety* reactor monitored execution to track a number of variables of interest for ensuring PR2 safety and enabled these variables to be shared by reactors. Note that it has a 0 lookahead, thus no goals will be dispatched to it. It simply derives estimates of safety state variables through the synchronization process and the domain model.

The Doorman state machine

Robust traversal of doorways proved a substantial challenge, where doorways could be in a variety of locked, closed, partially and fully open states. Fig. 6 illustrates the states and transitions for the door controller, which is planned and

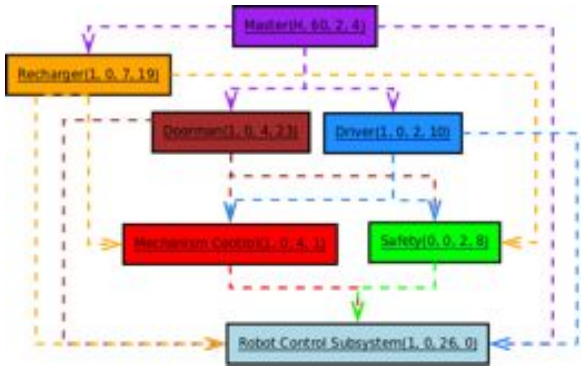


Figure 5: The reactor topology for PR2.

executed incrementally within the context of the *Doorman*. These states and transitions are hidden from other reactors. A state machine is a common and convenient way to specify a control strategy. In TREX, we merge this concept with temporal planning to decompose states into actions using explicit decomposition as well as implicit action generation through planning. One can think of the reactive generation of a successor state as producing a near term goal to plan and execute.

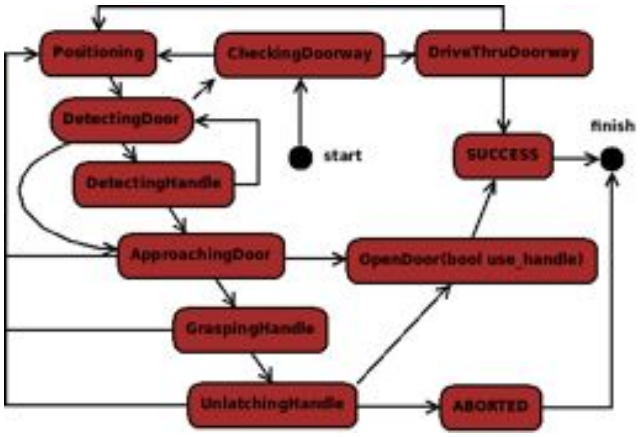


Figure 6: States and transitions for the door domain.

The state machine begins by checking for a clear path through the doorway. Space constraints preclude a detailed description of what occurs in each case. Suffice to say that each state maps to one or more external actions, and that successor states are specified based on the success or otherwise of any actions executed. Listing 3 shows a partial declaration of the door controller class that implements this state machine.

```
1 class DoorController extends StateMachine {
2   predicate Inactive{}
3   predicate DetectingDoor{}
4   predicate OpenDoor{ bool use_handle; }
```

Listing 3: NDDL declaration of the door controller class

The model fragment in Listing 4 provides a simple illustration of mapping a state to an action, and generating the successor state conditionally based on action feedback. The model indicates a subgoal that ensures a *detect_door* action executes within the temporal scope of this state, and links the end times of this state with the end time of the action. A *status* parameter of the action, which is only bound when the action completes and its feedback produces a result, generates the successor state. All requirements for this action dictated by the model must also be met.

```
1  DoorController::DetectingDoor{
2    contains(detect_door.Active cmd);
3    ends cmd;
4    if(cmd.status == SUCCESS){
5      cmd meets(detect_door.Inactive feedback);
6      if(feedback.latch_state == LATCH_STATE_UNLATCHED){
7        meets(ApproachingDoor);
8      } else {
9        meets(DetectingHandle);
10     }
11   } else {
12     meets(CheckingDoorway);
13   }
14 }
```

Listing 4: A decomposition rule for *DetectingDoor*

Concurrent and interacting actions

Not surprisingly, getting through the doorway while manipulating the door is the most involved aspect of doorway traversal. In particular, the decomposition of the *OpenDoor* state involves a number of actions with concurrency and ordering requirements. Listing 5 describes a model fragment expressing the decomposition for the case where the door is already unlatched (i.e. not using the handle). PR2 starts by touching the door, and then begins pushing. Once pushing begins, the robot can move the base. When the base reaches the goal, pushing may complete, but not before. Fig. 7 illustrates the resulting plan with appropriate temporal constraints.

```
1  DoorController::OpenDoor{
2    contains(move_base_door.Active cmd_mb);
3    cmd_mb before(stop_action.Active cmd_stop);
4    if(use_handle == false){
5      contains(touch_door.Active cmd_td);
6      cmd_td before(push_door.Active cmd_pd);
7      ends cmd_pd;
8      eq(cmd_td.status, SUCCESS);
9      cmd_pd starts_before cmd_mb;
10     eq(cmd_stop.action_name,"push_door");
11   }
12   if(cmd_mb.status == SUCCESS){
13     meets(Inactive s);
14     eq(s.status, SUCCESS);
15   } else{
16     meets(OpenDoor s);
17     eq(s.use_handle, false);
18   }
19 }
```

Listing 5: NDDL model fragment for the OpenDoor state

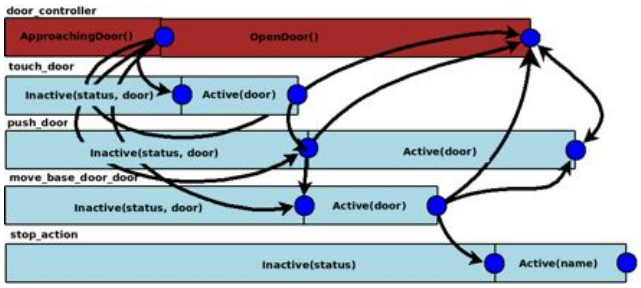


Figure 7: Tokens, timelines and temporal constraints for going through a doorway without a grasp on the door handle. The plan has not been executed yet.

Generative planning to apply the model

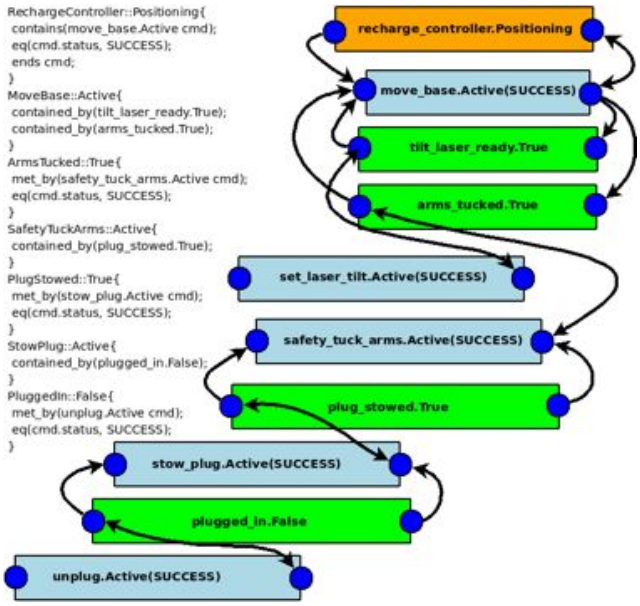


Figure 8: Subgoals generated by regression from the recharge controller state *Positioning*.

There are a number of constraints that apply to actions that are not specified directly as part of a state machine. For example, consider the simple case when the robot is positioning itself at an approach point for outlet detection. In order to move the base, the tilt laser must be ready and the arms must be tucked. These states are achieved by actions to set the tilt laser and tuck the arms respectively. Tucking the arms requires that the plug is stowed, which is achieved by stowing the plug. Finally, stowing the plug requires that the plug is unplugged, which is accomplished by the unplug action. This example is illustrated in Fig. 8. The relevant model elements are shown in text. Tokens are colored according to the reactor that owns them (Fig. 5). Lines with single arrows indicate a precedence constraint $\leq$, and double arrowed lines indicate an equality constraint. Note that if the conditions are satisfied at one step (e.g. the arm is already stowed), then no further subgoal generation occurs. The full plan is not shown. This example illustrates how a state machine for expressing control objectives is integrated with generative planning, greatly simplifying the specification of a single component since so many implied requirements can be handled automatically.

Results

Over the course of 3 weeks, PR2 and TREX were used to run different demonstrations using different target outlets, with doors and offices in different states. Notably, a critical demonstration required 10 recharge goals to be accomplished consecutively without any human intervention. This demonstration was accomplished in under an hour. In this and other demonstrations, TREX gracefully handled a wide range of failure conditions to succeed in each case. The data in this section is based on a single run given 9 recharge goals.

Measurement	**Value**
Executive control rate	10 Hz
Total number of *internal* timelines	47
Total number of *external* timelines	66
Total number of EUROPA timelines	87
Mission duration	3799 seconds
Total number of actions executed	494
Total number of action failures	29
Total number of planning cycles	907
Memory consumption for the executive	10 MB
Estimated model line count	1207
TREX CPU utilization (mean)	9.8%
TREX CPU utilization (std)	5.0%

Table 1: Summary metrics of complexity and performance

Table 1 provides a summary listing of key metrics. The executive operated at a control rate of 10 Hz. This provides snappy transitions from one action to another but requires synchronization of the complete executive once every 100 ms. The overall scale of the system is indicated by the number of internal and external timelines. The number of internal timelines reflects the total number of state variables referenced by the executive. This number includes 26 timelines that are internal to the *Robot Control Subsystem*. The total number of EUROPA timelines excludes these. The mission ran for just over an hour. 3 doors were locked when tried. The Executive deferred them till later. One of them was opened when revisited. The remaining 2 were continually retried until we terminated the test. In total, 494 external robot actions were executed, of which 29 aborted or timed-out. 907 planning cycles occurred across all deliberative reactors. A planning cycle is initiated when a reactor receives a goal, or when a flaw is entailed by the model. There were no plan failures. TREX memory consumption was flat at 10 MB. The total line count for the model provides a coarse metric of program complexity. It includes all constraints and class declarations. The modest number reflects the leverage

from automated planning and a very high level programming model.

Recall that TREX was running on a dual-core 2.6GHz Linux machine. CPU load is a key measure of the scalability of the system. The mean and standard deviations for CPU utilization are at 9.8% and 5.0% respectively. This indicates that even at a control rate of 10 Hz, TREX was comfortably able to handle the load. Time series data for CPU utilization over the full run is shown in Fig. 9. The baseline cost is under 10.0%, mainly due to synchronization of new pose observations at every tick, and propagating temporal constraints for all reactors. Increases arise where action transitions and planning occur, but remain well within our time budget.

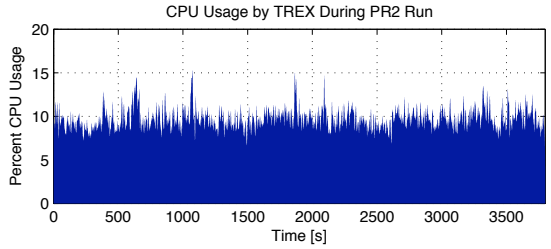


Figure 9: CPU utilization throughout execution

Discussion

The results obtained in this experiment were very encouraging, suggesting a promising pathway for deep integration of declarative models, and automated planning as a paradigm for practical robot programming. Not only did TREX play an important role in accomplishing the milestone, which was in itself a significant achievement in autonomous robotics, but it did so with modest computational overhead and system complexity. The combination of familiar state-machine specifications with generative planning proved very powerful and greatly simplified program complexity. We found the declarative temporal programming paradigm to be highly expressive, and very powerful, for capturing concurrency and ordering constraints among actions. The partitioning scheme was particularly important, allowing planning and synchronization costs to be localized on a need-to-know basis. While there might be large changes internal to the doorman, the driver and the master remain unaffected. Partioning also enabled reactors to be incrementally integrated and tested, and easily stubbed out. Notably, 5 of the 6 deliberative reactors used the same base configuration of the EUROPA planner. Only the master required specialized components to formulate and solve an orienteering problem for sequencing top-level goals. Despite a number of action failures, no replanning was required because of the incremental refinement of plans in the driver, doorman, and recharger based on action feedback.

Conclusions and Future Work

In order to aid adoption of TREX, we will focus on usability challenges, concentrating on improvements in the underlying language, and the tools used for monitoring and analysis.

Acknowledgements

We thank all resident and visiting engineers at Willow Garage for their hard work getting PR2 hardware and software to the point where we could conduct our experiments. We thank NASA and MBARI for making EUROPA and TREX available as open-source software. Finally, we thank Willow Garage for supporting this work.

References

Allen, J. F. 1983. Maintaining knowledge about temporal intervals. *Commun. ACM* 26(11):832–843.

Aschwanden, P.; Baskaran, V.; Bernardini, S.; Fry, C.; Moreno, M.; Muscettola, N.; Plaunt, C.; Rijsman, D.; and Tompkins, P. 2006. Model-unified planning and execution for distributed autonomous system control. In *Workshop on Spacecraft Autonomy, AAAI Fall Symposium*.

Beetz, M. 2000. Runtime plan adaptation in structured reactive controllers. In *in Proceedings of the Fourth ICAA*.

Bresina, J. L.; Jónsson, A. K.; Morris, P. H.; and Rajan, K. 2005. Activity planning for the mars exploration rovers. In *ICAPS*, 40–49.

Frank, J., and Jónsson, A. K. 2003. Constraint-based attribute and interval planning. *Constraints* 8(4):339–364.

Gat, E.; Bonnasso, R. P.; Murphy, R.; and Press, A. 1997. On three-layer architectures. In *Artificial Intelligence and Mobile Robots*, 195–210. AAAI Press.

McGann, C.; Py, F.; Rajan, K.; Ryan, J.; and Henthorn, R. 2008a. Adaptive control for autonomous underwater vehicles. In *AAAI*, 1319–1324.

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

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote agent: To boldly go where no ai system has gone before. *Artif. Intell.* 103(1-2):5–47.

Muscettola, N.; Dorais, G. A.; Fry, C.; Levinson, R.; and Plaunt, C. 2002. Idea: Planning at the core of autonomous reactive agents. In *in Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*.

Nilsson, N. J. 1994. Teleo-reactive programs for agent control. *J. Artif. Intell. Res. (JAIR)* 1:139–158.

Quigley, M.; Gerkey, B.; Conley, K.; Faust, J.; Foote, T.; Leibs, J.; Berger, E.; Wheeler, R.; and Ng, A. 2009. Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*.

Shanahan, M. 2000. *Reinventing shakey*. Norwell, MA, USA: Kluwer Academic Publishers.

Williams, B. C.; Ingham, M. D.; Chung, S. H.; and Elliott, P. H. 2003. Model-based programming of intelligent embedded systems and robotic space explorers. In *In Proceedings of the IEEE: Special Issue on Modeling and Design of Embedded Software*, 212–237.