retrou sur innovation

www.onera.fr
A generic framework for anytime execution-driven planning in robotics

Florent Teichteil-Königsbuch, Charles Lesire-Cabaniols, Guillaume Infantes

ICRA 2011 — Shanghai, China — Tuesday, May 10, 2011
Illustrative example: emergency autonomous landing

- unknown environment
- map a rectangular zone and quickly find a place to land
- candidate landing zones after automated mapping

- candidate zones not necessary landable!
- need for a long-term planning of candidate landing zones to explore in order to minimize the mission’s duration
- Which contingent strategy to apply depending on hazards?
Illustrative example: emergency autonomous landing

- Huge state space due to many state variables:
  - (on-ground)
  - (explored ?z - zone)
  - (landable ?z - zone)
  - (at ?z - (either base zone))
  - (com)
  - (fuel-level)
  - (available-memory)

- Modeled as a Markov Decision Process necessary solved on-line after image processing

- Worst-case optimization time with an embedded computer running at 2 Ghz (assuming on-board memory is sufficient): 55 minutes with 5 zones (540 years with 10 zones) but mission’s typical duration is about 15 minutes!

- Need for a (different) deterministic planner for generating exploration paths in candidate landing zones

- Need to formally validate the safety of the entire mission
Automated planning: definition

Automated planning is a branch of artificial intelligence concerning the automatic generation of strategies or action sequences that achieve a given objective knowing an initial state and actions effects.

Automated planning: features

- long-term and deliberative reasoning
- combinatorial explosion
- consumes memory and CPU time

Automated planning: challenges for robotics

- interaction with other functionalities (perception and action)
- real-time decisions
- validation of decisions w.r.t. the entire architecture
Automated planning: a generic formalism

- \( S \): set of states
- \( S_I \): set of initial states
- \( S_G \): set of goal states
- \( A \): set of actions
- \( O \): set of observations
- \( T : S \times A \rightarrow 2^S \): transition function
- \( R : S \times A \times S \rightarrow \mathbb{R} \): reward function
- \( O : S \times A \rightarrow 2^O \): observation function

Purpose: design a generic planning function based on the above concepts
A single planning component, with a variable planner

- Same interface for all planners
- Same behavior for all planners
- Behavior’s code independent from the planner used (classical, MDP, POMDP)
- Reasoning data structures owned by planners
- Facilitates reusability and validation
Basic concepts: planning request & action request

Planning request (plan construction)

- set of **initial states** from which the planner must compute an optimized action (knowing long-term requirements);
- **time allocated** to the plan construction;
- **algorithm** used to construct the plan;
- **algorithm parameters**.

Action request (plan execution)

An optimized action to apply in a given state.
Anytime property, planning & action request interleaving

plan from state $s_0$ during 10 s.

Initial planning phase from the initial state (bootstrap)
Anytime property, planning & action request interleaving

- Execute best action
- Plan from state $s_0$ during 5 s. (or default action)
- Keep $s_4$ and its potential successors as very likely reachable

// Execution of the best action planned in $s_0$, approximate execution time is 6 s.
// Planning from possible next states during 2 s. each.
Anytime property, planning & action request interleaving

// Execution of the best action planned in current state $s_2$, approximate execution time is 7 s.
// Planning from states of the most probable execution path.
Anytime property, planning & action request interleaving

- Model shift: state $s_1$ was actually reachable from $s_2$!
- Plan from current state $s_1$ during 5 s. (or default action)
- Keep $s_4$ and its potential successors as very likely reachable
On-line planning component: state machine

- Waiting Problem
- load_problem
- Loading Problem
- Problem Loaded
- Stopped
- Error
- Planning
- Problem Solved

- stop
- add_plan_request
- remove_plan_request
- get_action

- blocking
- non-blocking
- automatic transition when processing done
On-line planning component: requests management

- No need to assume the planner’s code is thread-safe
- Only the locally-copied policy $\tilde{\pi}$ is protected by mutex
- Default policy filtering action requests (validation & reactivity)
Variable planner as a template of the planning component

Each planner is a class that must define the following embedded types and methods.

class Planner {

    // Embedded types
    class problem_type {...};
    class state_type {...};
    class state_set_type {...};
    class action_type {...};
    class action_set_type {...};
    class policy_type {...};
    typedef enum {...} algorithm_enum;
    class algorithm_parameters_type {...};
    class algorithm_statistics_type {...};

    // Member functions
    void problem(const problem_type&);
    void load_problem_begin();
    void load_problem_progress();
    bool problem_loaded() const;
    void load_problem_end();
    void algorithm(algorithm_enum,
                   const algorithm_parameters_type&);

    void solve_begin(const state_set_type&);
    void solve_progress();
    void solve_end();
    bool converged() const;
    bool plan_defined(const state_type&) const;
    action_type get_action(const state_type&) const;
    action_type default_action(const state_type&) const;
    algorithm_statistics_type get_statistics() const;
    void update_policy(policy_type&,
                        const state_set_type&) const;
    static bool plan_defined(const policy_type&,
                              const state_type&);
    static action_type get_action(const policy_type&,
                                   const state_type&);
    static action_set_type get_actions(const state_type&);
    static state_set_type get_effects(const state_type&,
                                       const action_type&);
};
Search & rescue mission

Planning components used:

PlanningComponent<HMDPPlanner>
PlanningComponent<AstarPlanner>
Random problems: planning time against mission time

‘off’: plan-then-execute approach  ‘on’: our approach

Average proportion of planning time against total mission time (among missions)

Planning process
Other processes

Total mission time (until success)

<table>
<thead>
<tr>
<th>Planning problem size (mission duration: 10 mn)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
</tr>
</tbody>
</table>

Off     | On     | Off     | On     | Off     | On     | Off     | On     | Off     | On     |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rtdp</td>
<td>rtdp</td>
<td>rtdp</td>
<td>rtdp</td>
<td>rtdp</td>
<td>rtdp</td>
<td>rtdp</td>
<td>rtdp</td>
<td>rtdp</td>
<td>rtdp</td>
</tr>
</tbody>
</table>

F. Teichteil, C. Lesire, G. Infantes – A generic framework for anytime execution-driven planning in robotics
Random problems: plan length until reaching the goal

‘off’: plan-then-execute approach

‘on’: our approach

Average plan length until reaching the mission goal (among missions)

Plan length (until success)

Planning problem size (mission duration: 10 mn)

100
1000
10000
100000
250000

2
3
4
5
6
7
8
9
10
Random problems: successes, timeouts, default actions

‘off’: plan-then-execute approach

‘on’: our approach

Percentages of success, time-outs and default actions (among missions)

Mission duration (problem size: 100000)

Percentages

success

time-outs

default actions

10 mn
1 mn
30 s

Mission duration
(problem size: 100000)
Conclusion and perspectives

- Design of a **generic and reactive planning component** for a modular robotics architecture
- Provide *immediate* services on demand to other modules
- Separation between requests’ management (component) and planning algorithms (planner)
  - **same requests’ management for all planners**
  - **planners are (template) plugins of the component**
- Implementation on the Orocos platform
- Experiments on a high dimensional search & rescue mission, and random challenging benchmarks
- Close future: Validate the planning components’ behavior
  - Validate the component (requests’ management) once and for all, assuming satisfied properties on the planner side
  - Validate all planners plugged to the planning component
  - Validate the default policy for each mission
Questions?

Thank you for your attention :-)

[Image of a helicopter and monitoring screen]