ONERA

THE FRENCH AEROSPACE LAB

retour sur innovation

www.onera.fr

A generic framework for anytime execution-driven planning in robotics

Florent Teichteil-Königsbuch, Charles Lesire, Guillaume Infantes

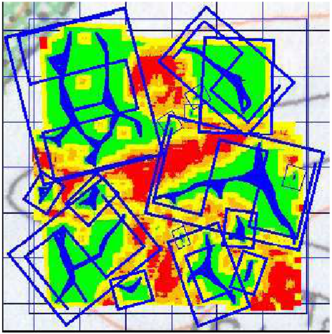
CAR 2011 — Grenoble, France — May 2011

ONERA
THE FRENCH AEROSPACE LAB

retour sur innovation

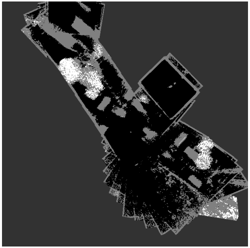
Illustrative example: autonomous emergency landing

- 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: autonomous emergency 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! **reactivity**
- Need for a (different) deterministic planner for generating exploration paths in candidate landing zones **genericity**
- Need to formally validate the safety of the entire mission **validation**

Automated planning: definition

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**

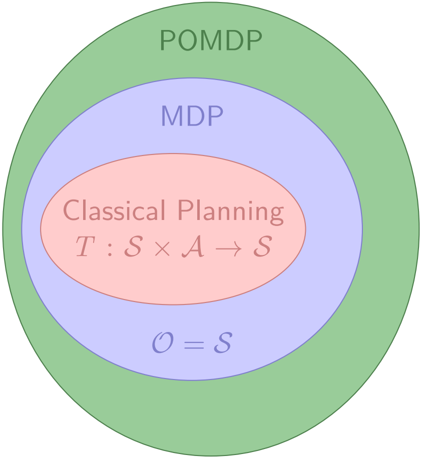
- long-term and deliberative reasoning
- combinatorial explosion
- consumes memory and CPU time

Automated planning: **challenges for robotics**

- interaction with other functionalities (perception and action)
- real-time decisions
- validation of decisions w.r.t. the entire architecture

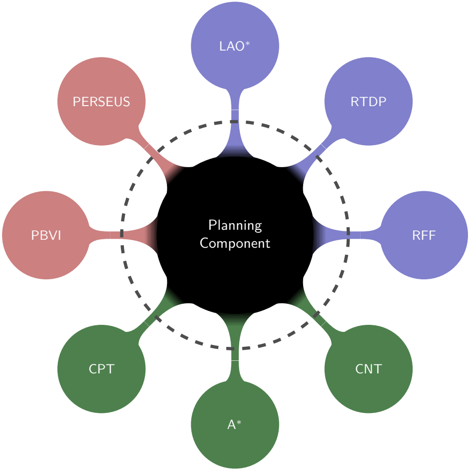
Automated planning: a generic formalism

- $\mathcal{S}$: set of states
- $\mathcal{S}_{\mathcal{I}}$: set of initial states
- $\mathcal{S}_{\mathcal{G}}$: set of goal states
- $\mathcal{A}$: set of actions
- $\mathcal{O}$: set of observations
- $T : \mathcal{S} \times \mathcal{A} \rightarrow 2^{\mathcal{S}}$: transition function
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$: reward function
- $O : \mathcal{S} \times \mathcal{A} \rightarrow 2^{\mathcal{O}}$: 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**

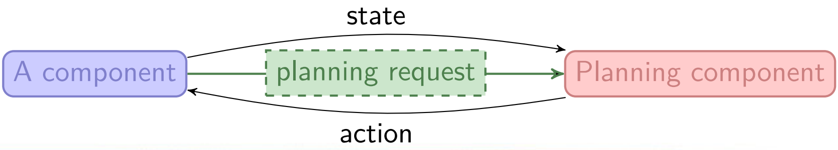
Basic concepts: planning request & action request

Planning request (plan construction)

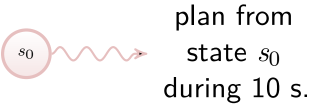
- set of **initial states** from which the planner must compute an optimized action (knowing long-term requirements) ;
- **time allocated** to the plan construction ;
- **algorithm** used to construct the plan ;
- algorithm **parameters**.

Action request (plan execution)

An optimized action to apply in a given state.

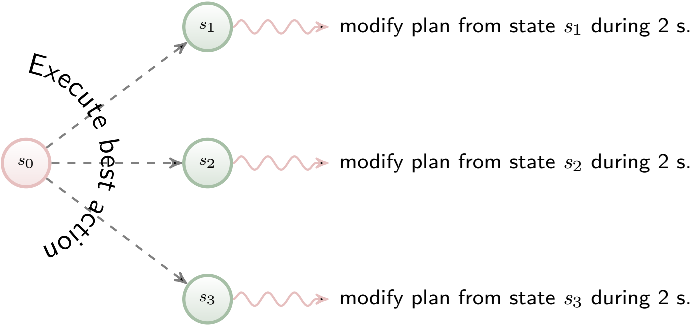


Anytime property, planning & action request interleaving



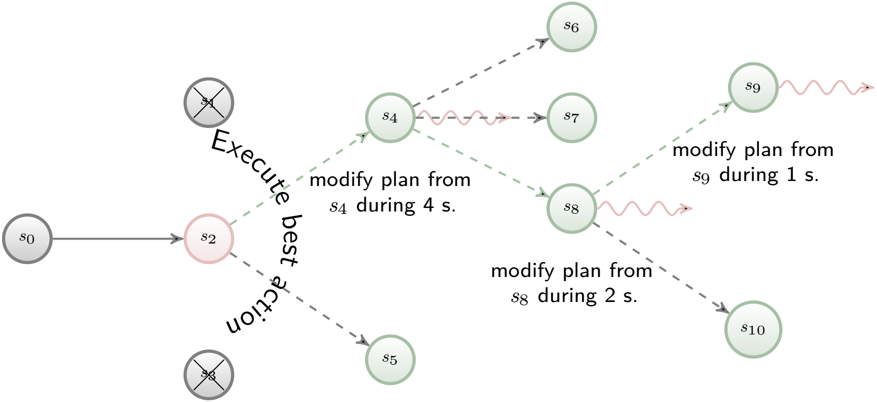
Initial planning phase from the initial state (bootstrap)

Anytime property, planning & action request interleaving



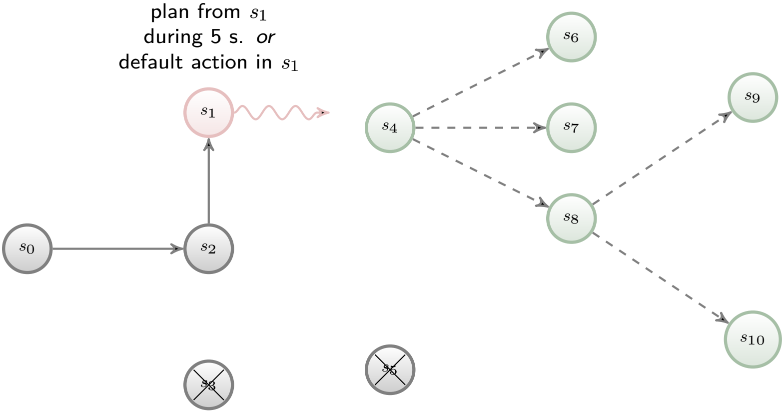
// Execution of the best action planned in s_0, approximate execution time is 6 s.
// Planning from possible next states during 2 s. each.

Anytime property, planning & action request interleaving



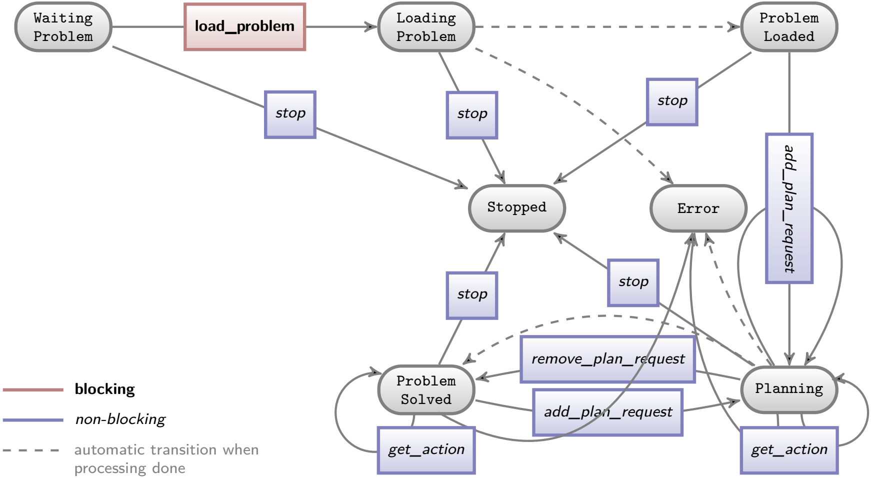
// Execution of the best action planned in current state s_2, approximate execution time is 7 s.
// Planning from states of the most probable execution path.

Anytime property, planning & action request interleaving

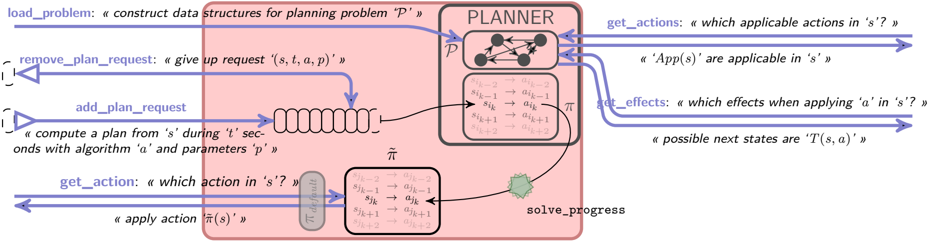


- Model shift: state s_1 was actually reachable from s_2!
- Plan from current state s_1 during 5 s. (or default action)
- Keep s_4 and its potential successors as very likely reachable

On-line planning component: state machine



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

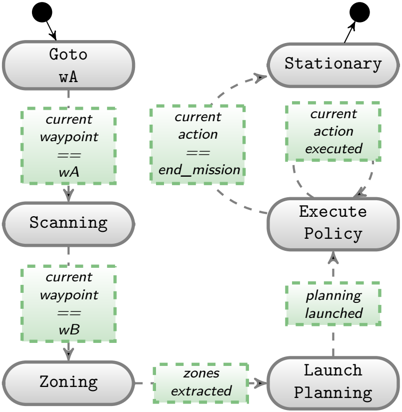
Each planner is a class that must define the following embedded types and methods.

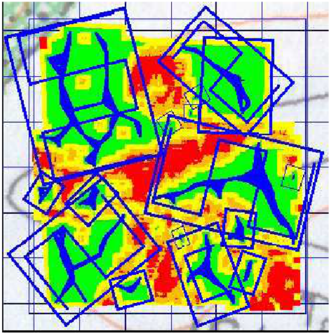
```
class Planner {

// 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&);
action_set_type get_actions(const state_type&) const;
state_set_type get_effects(const state_type&,
               const action_type&) const;
};
```

Search & rescue mission





Zones extracted after

Scanning + Zoning

Planning components used: PlanningComponent<HMDPPlanner>
PlanningComponent<AstarPlanner>

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)
  $\Rightarrow$ **same requests' management for all planners**
  $\Rightarrow$ **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 :-)