

Continuous mission plan adaptation for autonomous vehicles: balancing effort and reward

Pedro Patrón, David M. Lane, Yvan R. Petillot

Ocean Systems Laboratory, Heriot-Watt University*

EH14 4AS, Edinburgh, Scotland

{p.patron,d.m.lane,y.r.petillot}@hw.ac.uk

Abstract

This paper proposes a novel approach for adaptive mission planning for autonomous vehicles in a changing discoverable environment. The approach handles temporal planning with durative actions, metric planning, opportunistic planning and dynamic planning. During the planning process, plan selection is balanced between its estimated cost of execution and the reward obtained by reaching the new configuration of the environment. The plan proximity between plans is defined in order to measure difference between plans performing on the same execution environment. The approach is evaluated under a static scenario and a partially known dynamic scenario using the plan proximity to the human driven mission.

Introduction

Robotic platforms are helping humans to gain routine and permanent access to different environments. However, challenges related to difficult domains, such as underwater, require the integration of embedded tools that can raise the platform's autonomy levels and adapt to the continuous changes on the perception that the platform has of the environment.

The problem, however, is that at present, applications are mono-domain: mission targets are simply mono-platform, and missions are generally static procedural lists of commands (Hagen 2001). If behaviours are added (Pang et al. 2003), they are only to cope with possible changes that are known *a-priori* by the operator. All this, therefore, leaves the platforms in isolation and limits the potential of adapting mission plans to the sensing information. The benefits of autonomous dynamic mission planning in changing domains for has been promoted by Rajan et al. in (Rajan et al. 2007) and (McGann et al. 2007).

We propose an approach based on a continuous re-assessment of the status of the environment resources and the platform capabilities. The approach combines a Bayes model for prediction, measurement and correction (Thurn, Burgard, and Fox 2005) with the classical

planning approach. Instead of solving a plan from initial state to goals like in classical AI planning, the approach tries to maintain a window of actions that are believed can be performed from of the current state in order to improve a given utility function.

The paper is structured as follows: The following section describes the modeling of the environment of execution or domain. Next, we introduce the way the problem or mission gets represented. The proposed plan strategy to solve this problem is described in the third section. A measure of plan proximity is proposed for comparing different planning strategy results. Based on this metric, the final section analyses the performance of the approach in a static and in a partially known dynamic scenario. The paper ends with the conclusions and the future work.

Domain Model

We assume that the planner has access to the domain knowledge describing the list of actions/capabilities and resources.

This domain is defined by the tuple $\Sigma = (C, O_C, V_C, P_V, A_V)$, where:

- $C = \{c_i | i \in \langle 1, 2, \dots, |C| \rangle\}$ ¹ is a set of hierarchical classes of objects. All classes derive from a root class named `object` (e.g.: `(:types location - object area - location) class(c)` represents the set form by class c and all its ancestors:

$$class(c) = \begin{cases} \{\text{object}\}, & \text{if } c=\text{object} \\ \{c\} \cup \{class(parent(c))\}, & \text{otherwise} \end{cases}$$

- $O_C = \{o_j^{c_i} | j \in \langle 1, 2, \dots, |O_C| \rangle, c_i \in C\}$ represents a set of objects or resources of C . Each object o_j is of only one class c_i . Therefore each object o_j belongs to its class and all the ancestors of this class ($\forall o_j^{c_i} \in O_C \ class(o_j) = class(c_i)$). (e.g.: `seabedA - area` indicates that the region 'seabedA' belongs to the `area`, `location` and `object` classes).
- $V_C = \{v_k^{c_i} | k \in \langle 1, 2, \dots, |V_C| \rangle, c_i \in C\}$ is a set of variables of C . In the same way, each variable v_k belongs to its class and all the ancestors of this

*This paper is partly funded by the Project RT/COM/5/059 from the Competition of Ideas and by the Project SEAS-DTC-AA-012 from the Systems Engineering for Autonomous Systems Defence Technology Centre, both established by the UK Ministry of Defence.

¹A set is a collection of elements. It is represented by $\{\}$. A list is an ordered set. It is represented by $\langle \rangle$.

class $(\forall v_k^{c_i} \in V_C \text{ class}(v_k) = \text{class}(c_i))$. (e.g.: `?v - vehicle` indicates a variable v of the class `vehicle`). An ordered set of variables and objects defines a list of arguments for an item x :

$$\text{arg}(x) = \left\langle v_k^{c_i} | k \in \langle 1, 2, \dots, n \rangle, \right. \\ \left. 0 \leq n \leq |V_C| + |O_C|, c_i \in C \right\rangle \\ \subseteq \{V_C \cup O_C\}$$

- $P_V = \{p_m | m \in \langle 1, 2, \dots, |P_V| \rangle\}$ is a set of propositions. A proposition can return a boolean or a numeric value. Each proposition p_m has a list of arguments $\text{arg}(p_m)$. (e.g.: `(at ?l - location)` represents the proposition of being at a particular location).

$F_V = \{f_q | q \in \langle 1, 2, \dots, |F_V| \rangle\} \subseteq P_V$ is a set of functions. A function is a proposition that returns a numeric value. (e.g.: `(distance ?a ?b - location)` represents the value of the distance between two locations).

- $A_V = \{a_h | h \in \langle 1, 2, \dots, |A_V| \rangle\}$ is a set of actions. Each action a_h has a list arguments $\text{arg}(a_h)$. An action can have a set of requirements: $\text{condition}(a_h) = \{p_m | p_m \in P_V \wedge \text{arg}(p_m) \subseteq \text{arg}(a_h)\}$. An action can have a set of effects: $\text{effect}(a_h) = \{p_m | p_m \in P_V \wedge \text{arg}(p_m) \subseteq \text{arg}(a_h)\}$. An action has some duration in time: $\text{duration}(a_h) \in \mathbb{R}$. (e.g.: `(move (?from ?to - location) (:dur (distance ?from ?to)) (:cond (at ?from)) (:effect (not (at ?from)) (at ?to)))`).

From this tuple, another two sets can be calculated:

- $R_O = \{r_y^{p_m} | y \in \langle 1, 2, \dots, |R_O| \rangle, p_m \in P_V, \text{arg}(r_y^{p_m}) \subseteq O_C\}$ is the set of proposition facts. A proposition fact $r_y^{p_m}$ is an instantiation of a proposition p_m for a particular list of objects as arguments $\text{arg}(r_y^{p_m})$. (e.g.: `(at seabedA)`).
- $G_O = \{g_z^{a_h} | z \in \langle 1, 2, \dots, |G_O| \rangle, a_h \in A_V, \text{arg}(g_z^{a_h}) \subseteq O_C\}$ is the set of ground actions. A ground action g_z is an instantiation of an action a_h for a particular list of objects as arguments $\text{arg}(g_z^{a_h})$. (e.g.: `(move start seabedA)`).

An action can be probabilistic. Given an action a_h with probabilistic effects, the uncertainty matrix Γ for this action can be defined as:

$$\Gamma(a_h)_{\text{effect}(a_h) \times |R_O|} = \left\{ p(i|j) \right. \\ \left. i \in \text{effect}(a_h), j \in R_O \right\}$$

Problem Model

$t \in \mathbb{R}$ defines the continuous time of the mission. $s \in \mathbb{N}_0$ defines a discrete step or slot in time of a certain duration $d_s : [t_0^s, t_n^s]$.

A state at some particular step in time x_s is a set containing information of the available actions, available resources and the combination of possible proposition facts at that step:

$$x_s = x_s^{A_V} \cup x_s^{O_C} \cup x_s^{R_O} \\ = \{i(x) \in [0; 1] \forall x \in \{A_V \cup O_C \cup R_O\}\}$$

It can be seen that $|x_s| = |A_V| + |O_C| + |R_O|$.

A ground action $g_s^{a_h}$ at step s defines a transition function between states $g_s^{a_h} : x_{s-1} \rightarrow x_s$ through the sequence of steps.

A plan u_s^T defines a list of ground actions to be performed in the T steps ahead $u_s^T : x_{s-1} \rightarrow \langle g_s, g_{s+1}, \dots, g_n | n \leq T \rangle$, where T defines the number of ground actions (windows size) to be included in the continuous plan. T is also known as the *planning horizon*. e_s^t defines the execution of g_s at time t .

Rewards

Each object $o_j^{c_i} \in O_C$ has a reward value $\delta(o_j^{c_i})$.

The reward of a proposition fact $r_y^{p_m} \in R_O$ is the sum of all the rewards of the objects used as arguments:

$$\delta(r_y^{p_m}) = \sum \delta(o_j^{c_i}) | \forall o_j^{c_i} \in \text{arg}(r_y^{p_m})$$

The reward of a state is the sum of all the rewards of the proposition facts available on it:

$$\delta(x_s) = \sum \delta(r_y^{p_m}) | (r_y^{p_m}) | (r_y^{p_m}) \in x_s^{R_O}$$

The set of mission goals can be defined explicitly with a list of rewards assigned to different proposition facts.

- $Q_O = \{\lambda(r_y^{p_m}) | r_y^{p_m} \in R_O, \text{arg}(r_y^{p_m}) \subseteq O_C\}$ is the set of proposition fact goals. $\lambda(r_y^{p_m})$ is a reward defined by the operator. (e.g.: `(= (surveyed seabedA) 100)`).

The reward of a ground action $g_s^{a_h}$ is related to the production of a proposition fact that has been explicitly defined as a mission goal in the mission problem. This means that the ground action has to produce a goal proposition fact in the new state that was not available in the previous state:

$$\delta(g_s^{a_h}, x_{s-1}) = \sum \lambda(r_y^{p_m}) \\ | r_y^{p_m} \in \{x_s^{R_O} \cap Q_O\} \\ \wedge r_y^{p_m} \notin \{x_{s-1}^{R_O} \cap Q_O\}$$

If a_h has probabilistic effects, the reward of the ground action $g_s^{a_h}$ is related to the probabilistic increase in the production of mission goals in the mission problem:

$$\delta(g_s^{a_h}, x_{s-1}) = \sum \lambda(r_y^{p_m}) \times \\ (\Gamma(a_h)[r_y^{p_m} | x_{s-1}] - x_{s-1}^{R_O}[r_y^{p_m}]) \\ | r_y^{p_m} \in Q_O$$

Costs

Each ground action $g_s^{a_h}$ has an execution cost when being executed in a state $\gamma(g_s^{a_h}, x_{s-1})$.

Payoffs

The *payoff* function σ of a ground action $g_s^{a_h}$ in a state x_{s-1} is the difference between its cost and the rewards of the proposition facts of the generated state x_s :

$$\sigma(g_s^{a_h}, x_{s-1}) = \delta(x_s) \\ + \delta(g_s^{a_h}, x_{s-1}) \\ - \gamma(g_s^{a_h}, x_{s-1})$$

The cumulative payoff of a plan u_s of length T given a state x_s is the expected utility function of the plan at that state $\sigma(u_s^T, x_s)$. It is the difference between the rewards accumulated through all the expected states and the ground actions in the plan:

$$\widehat{\sigma}(u_s^T, x_{s-1}) = E\left[\sum_{\tau=1}^T \beta^\tau \sigma(g_{s+\tau}^{a_h}, x_{s+\tau-1})\right]$$

where $\beta \in [0; 1]$ is known as the *discount factor*. It represents the fact that actions that are planned in the long term may have less effect over the current state than short term actions.

Passive Action

An action should always exist called **passive-action** (ϕ). This action has no preconditions, no effects and a unitary cost ($\forall s, \gamma(g_s^\phi, x_{s-1}) = 1$).

Solution Approach

The approach assumes that planning, observation and execution are performed concurrently. Under such assumption, a plan is calculated up to a defined planning horizon. While executing this plan, the environment is continuously observed. When all the plan is executed, the plan is calculated. If the environment (resources and/or capabilities) changes, the approach supports a greedy and a lazy behaviour. Under the lazy behaviour, planning is only performed at the end of the current plan execution ignoring changes on the environment. The greedy approach recalculates the plan as soon as the changes have been detected. The pseudo-code describing this process can be seen in Fig. 7.

We assume a framework of stochastic environments with fully observable states. This is known as Markov decision processes. A *policy* in this framework is a mapping from states to plans $\pi : x \rightarrow u$. In this framework the current state is sufficient for determining the optimal control. A policy selects the plan u_s^τ of horizon τ that maximizes the expected cumulative payoff from the current state x_{s-1} :

$$\pi_s^\tau(x_{s-1}) = \operatorname{argmax}_u [\sigma(u_s^\tau, x_{s-1})] \mid 1 \leq \tau \leq T$$

This policy is implemented using exhaustive planning search in the state-space. This policy is described in Fig. 8.

Given a plan policy π_s^τ , the *plan matrix* $\widehat{\Delta}_s^\tau$ contains information of the expected actions and resources used by the plan ahead. This matrix has T rows and $|A_V| + |O_C|$ columns:

$$\widehat{\Delta}_s^\tau = [\mu]_{T \times (|A_V| + |O_C|)} \mid \mu \in [0; 1]$$

Given a row $\varsigma \leq \tau$ and an action a_h ,

$$\mu_{\varsigma, a_h} = \begin{cases} 1, & \text{if } g_{s+\varsigma} \in \pi_s^\tau(x_{s-1}) \\ 0, & \text{otherwise} \end{cases}$$

Given a row ς and an object o_j ,

$$\mu_{\varsigma, o_j} = \begin{cases} 1, & \text{if } o_j \in \widehat{x}_{s+\varsigma} \\ 0, & \text{otherwise} \end{cases}$$

Action Management

- *Removal of an action that is not in plan:* In this case the state is corrected and the execution continues.
- *Removal of an action that is in the plan:* In this case the state and the plan are corrected.
- *Action recovery:* In this case, an action/capability that was previously available is recovered. The state gets corrected, and the plan is not guaranteed to be optimal for the window if a lazy approach is used. A lazy approach means that the plan only gets recalculated when the plan is empty.
- *New action:* When a new action is inserted in the system, the state and plan need to be corrected.

Object Management

- *Removal of an object that is part of the current state:* In this case the framework becomes unstable as the system no longer has available an object that was being used.
- *Removal of an object used in the plan:* In this case, the states need to get corrected and the plan recalculated.
- *Other removal of objects:* In this case, only the state needs correction.
- *Object recovery:* In a similar way as for the action recovery, when an object that was not available previously is recovered the state needs to get corrected. The plan is not guaranteed to be optimal for the window if a lazy approach is used.
- *New object:* When a new object is added, the state and the plan need to be corrected.

Predicate Management or Explicit Goals

Predicates can be managed through the use of goals. Goals are predicate facts that the operator wants them to happen.

Action with Probabilistic Effects

Actions can have probabilistic effects. In this case, the information of the state vector is probabilistic. The reward of the proposition facts in the goals is affected by the probabilistic effect of the actions.

Information Exchange

It can be seen that information about the current availability of actions and resources is stored in a single binary state vector. This vector can be easily compressed and transferred using low bandwidth communication hardware such as acoustic modems.

Plan Proximity

Plan proximity measures the similarity between two plans. Plan proximity can be calculated from the plan difference and the state difference of the outcome states (Patrón and Birch 2009).

The plan difference between u_1 and u_2 , $D_p(u_1, u_2)$ is the number of missing actions m_p from the reference plan u_1 and the number of extra actions e_p from a test plan u_2 that do not appear in the longest common subsequence of actions (Hunt and McIlroy 1976). The plan difference is normalized using the sum of the number of actions of the reference plan n_1 and the test plan n_2 .

$$\hat{D}_p(u_1, u_2) = \frac{m_p + e_p}{n_1 + n_2} \in [0; 1]$$

The state difference can be calculated as the Hamming distance (Hamming 1950) between the string representation of s_1 and s_2 . The state difference is normalized using the string length m .

$$\hat{D}_s(s_1, s_2) = \frac{\sum_{i=1}^m x_i}{m} \text{ where } x_i = \begin{cases} 0 & \text{if } s_1(i) = s_2(i) \\ 1 & \text{otherwise} \end{cases}$$

Plan proximity, $PP(u_1, u_2)$, is defined as the normalized balanced sum of the plan difference $D_p(u_1, u_2)$ and the state difference of the estimated final states that they are expected to produce $D_s(G_1, G_2)$.

$$PP_\alpha(u_1, u_2) = 1 - \alpha \cdot \hat{D}_p(u_1, u_2) - (1 - \alpha) \cdot \hat{D}_s(G_1, G_2) \in [0; 1]$$

where $\alpha \in [0; 1]$ represents a *balance factor* between plan and state difference.

Plan proximity is more informed than plan stability (Fox et al. 2006) for measuring plan strategies solving the dynamic planning problem as it takes into account actions missing from the reference plan, extra actions added in the test plan, sequential ordering of the plans and the expected outcomes states of these plans.

Evaluation

Using this metric, the approach has been evaluated under the scenario described by the Student Autonomous Underwater Challenge - Europe mission rules (SAUCE 2009). For this competition the scenario is form by three aligned gates, a bottom target, a moving mid-water target, two wall sections and a docking station. The vehicle should pass through three aligned gates, first forward and then backwards. Green and red lights on the second gate indicate the route that should be taken for its avoidance during the forward pass. After passing the gates in and out, the vehicle should attempt (in no particular order) to perform an inspection of the bottom target, to follow the moving mid-water target, to survey the walls and to dock at the docking station. The scenario is represented in Fig. 1.

The scenario is described using four files based on the PDDL syntax (Ghallab et al. 1998):

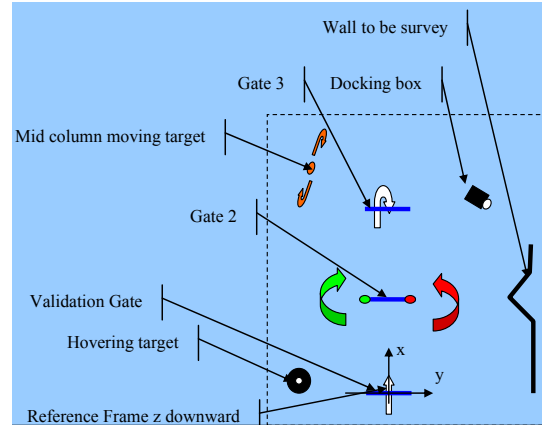


Figure 1: Scenario of the 2009 Student Autonomous Underwater Challenge - Europe. Mission starting at the origin. Gates, bottom target, mid-water target, wall sections and docking station locations are shown.

- The *domain*: describes the classes in C , constants of O_C , the predicates P_V , the functions F_V and the actions A_V .
- The *problem*: describes the initial state x_0 , the rewards $\delta(O_C)$, the goals Q_O and the cost metric γ .
- The *world model*: describes the known objects in O_C and their particular domain properties. These properties allow the calculation of the functions in F_V .
- The *dynamic world model*: simulates events that occur on the execution time line. Events can be triggered by a predicate in the current state or by reaching a slot. They can add, delete and restore objects, actions and predicates from the current knowledge.

Known Static Environment

This section analyses the outcomes of the proposed planning strategies in a fully known static environment. In this case, the world model contains all the information about the environment and the dynamic world file is empty.

The different planning strategies are evaluated looking at their plan proximity to u_0 (see Fig.2) after removing any instances of the passive action ϕ that may appear on the plan. Each approach was executed until $t > 220$ seconds. Fig.3 describes the cumulative payoff for different planning strategies. Table 1 shows plan proximity to u_0 of different planning strategies. $T \in 2, 3, 4$ generates the same plan as the human. $T = 1$ does not have enough evidence about rewards to commit to different actions other than the passive action. $T = 5$ reaches the exhaustive search limits and stops a step before providing the final section of the plan.

Partially-known Dynamic Environment

This section analyses the outcomes of the proposed planning strategies when solving the dynamic planning problem under the same scenario. In this case, the original world model only contains information about the

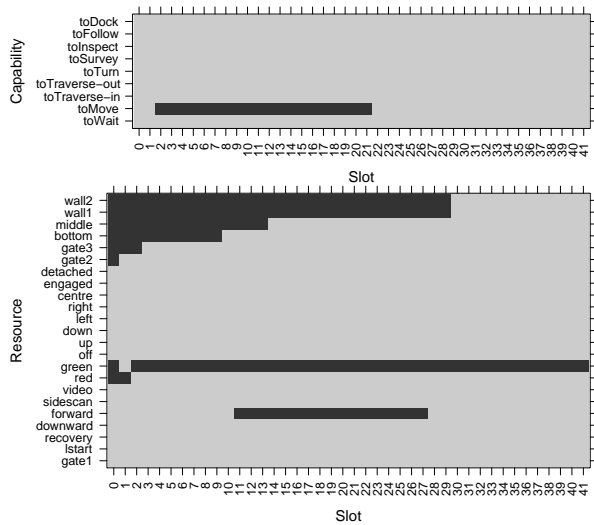


Figure 4: Evolution of capabilities and resources over time for the human driven mission (dark grey means unavailable). `gate2`, `gate3`, `bottom`, `middle`, `wall1` and `wall2` are discovered during the mission. `gate2` lights are discovered and change colour during the mission. `forward` camera and action `toMove` are temporarily unavailable during the mission.

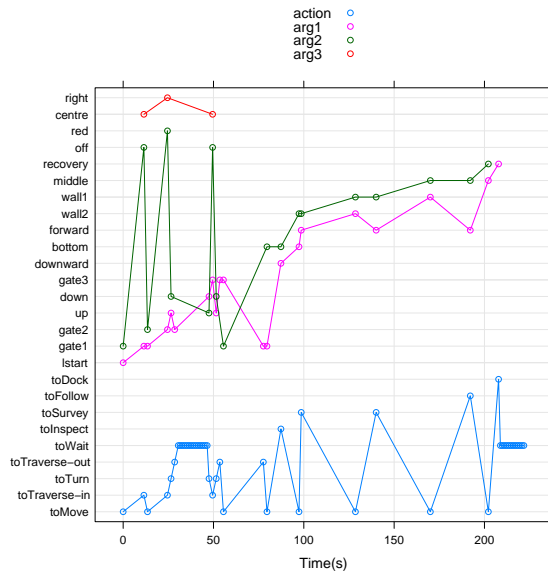


Figure 5: Ground actions with their arguments executed over time for the human driven mission in the dynamic environment scenario.

Technical report, Yale Center for Computational Vision and Control.

Hagen, P. 2001. Auv/uuv mission planning and real time control with the hugin operator system. In *IEEE OCEANS 2001*, volume 1, 468–473.

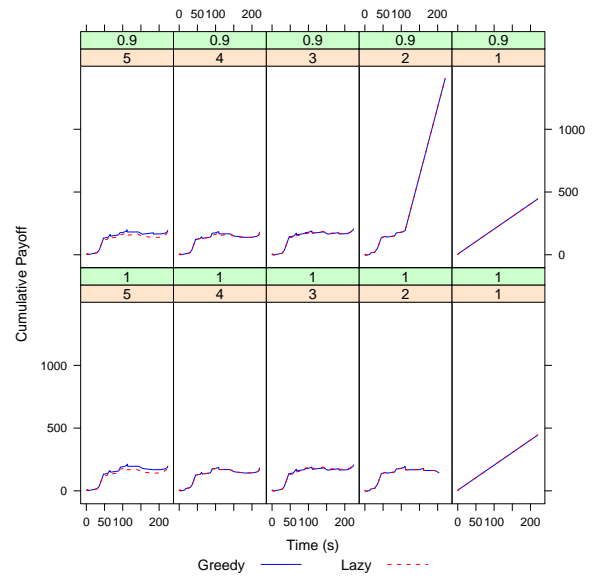


Figure 6: Cumulative payoff of different planning strategy results solving the partially known dynamic scenario. $T \in [1; 5]$, $\beta \in [0.9; 1]$ and $lazy \in [0; 1]$.

Hamming, R. W. 1950. Error detecting and error correcting codes. *Bell System Technical Journal* 29 2:147–160.

Hunt, J. W., and McIlroy, M. D. 1976. An algorithm for differential file comparison. *Computing Science Technical Report, Bell Laboratories* 41.

McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2007. T-rex: A model-based architecture for auv control. In *Workshop in Planning and Plan Execution for Real-World Systems: Principles and Practices for Planning in Execution, International Conference of Autonomous Planning and Scheduling*.

Pang, S.; Farrell, J.; Arrieta, R.; and Li, W. 2003. Auv reactive planning: deepest point. In *IEEE OCEANS 2003*, volume 4, 2222–2226.

Patrón, P., and Birch, A. 2009. Plan proximity: an enhanced metric for plan stability. In *Workshop on Verification and Validation of Planning and Scheduling Systems, 19th International Conference on Automated Planning and Scheduling*.

Rajan, K.; McGann, C.; Py, F.; and Thomas, H. 2007. Robust mission planning using deliberative autonomy for autonomous underwater vehicles. In *ICRA Robotics in challenging and hazardous environments*.

SAUCE. 2009. Mission rules for the student autonomous underwater challenge 2009 - europe v.1. Technical report, Defence Science and Technology Laboratory - Ministry of Defence, UK.

Thurn, S.; Burgard, W.; and Fox, D. 2005. *Probabilistic robotics*. MIT Press.

```

t ← s ← 0
T ← τ ← planning_horizon
x_s ← x_s^{Av} ∪ x_s^{Oc} ∪ x_s^{Ro}
greedy ← TRUE ∨ FALSE
Δ̂_s^τ ← zero_{T × (|Av| + |Oc|)}
s ← s + 1
forever do
  ;; adaptation
  τ ← T
  (π_s^τ, Δ̂_s^τ) ← argmax_u [σ(u_s^τ, x_{s-1})]
  recalculate ← FALSE
  while (recalculate ≠ TRUE) do
    ;; execute first action in the plan
    e_s^t ← g_0(π_s^τ)
    ;; predict next state
    (x̂_{s+1}, Δ̂_{s+1}^{τ-1}) ← exec(e_s^t)
    ;; observe state
    x'_{s+1} ← world_model
    ;; diagnosis
    if (|x̂_{s+1}| < |x'_{s+1}|)
      recalculate ← TRUE
    else
      if (x̂_{s+1} ⊃ x'_{s+1}) ∧ (Δ̂_{s+1}^{τ-1} ⊈ x'_{s+1} ∪ x'_{s+1}^{Oc} | ∀ς ≤ τ - 1) then
        if (Δ̂_{s+1}^{τ-1}(e_s^t) ⊈ x'_{s+1} ∪ x'_{s+1}^{Oc}) then
          unstable, execute emergency script!!!
        endif
        recalculate ← TRUE
      endif
      if (x̂_{s+1} ⊂ x'_{s+1}) ∧ (Δ̂_{s+1}^{τ-1} ⊆ x'_{s+1} ∪ x'_{s+1}^{Oc} | ∀ς ≤ τ - 1) ∧ (greedy) then
        recalculate ← TRUE
      endif
    endif
    ;; correction
    x_{s+1} ← x'_{s+1}
    t ← t + duration(e_s^t)
    τ ← τ - 1
    s ← s + 1
    if (τ = 0) then
      recalculate ← TRUE
    endif
  endif
endwhile
endfor

```

Figure 7: Approach in Pseudo-code

```

function SearchPolicy( $x_s, T$ )
  ;; initialize policy to the laziest plan: a sequence of pasive actions
   $\pi_s^T \leftarrow \{\phi_1, \phi_2, \dots, \phi_T\}$ 
   $\widehat{\sigma}(\pi_s^T, x_s) \leftarrow \text{goals}(x_s) + T \times \delta(x_s)$ 
   $\widehat{\Delta}_s^T \leftarrow \text{zero}_{T \times |A_V| + |O_C|}$ 
  ;; initialize search variables
   $\tau \leftarrow 0$ 
   $\mu_s^\tau \leftarrow \{\}$ 
   $\widehat{\sigma}(\mu_s^\tau, x_s) \leftarrow 0$ 
   $\widehat{\Delta}_s^\tau \leftarrow \text{zero}_{T \times |A_V| + |O_C|}$ 
  ;; launch exhaustive search
  ExhaustiveSearch( $\pi_s^T, \widehat{\sigma}(\pi_s^T, x_s), \widehat{\Delta}_s^T, x_s, \tau, \mu_s^\tau, \widehat{\sigma}(\mu_s^\tau, x_s), \widehat{\Delta}_s^\tau$ )
return ( $\pi_s^T, \widehat{\sigma}(\pi_s^T, x_s), \widehat{\Delta}_s^T$ )
endfunction

function ExhaustiveSearch( $\pi_s^T, \widehat{\sigma}(\pi_s^T, x_s), \widehat{\Delta}_s^T, \widehat{x}_{s+\tau}, \tau, \mu_s^\tau, \widehat{\sigma}(\mu_s^\tau, x_s), \widehat{\Delta}_s^\tau$ )
  ;; if the planning horizon is reached
  if ( $\tau = T$ ) then
    ;; if the plan found is better than the current one
    if ( $\widehat{\sigma}(\mu_s^\tau, x_s) > \widehat{\sigma}(\pi_s^T, x_s)$ ) then
      ;; adopt new plan
       $\pi_s^T \leftarrow \mu_s^\tau$ 
       $\widehat{\sigma}(\pi_s^T, x_s) \leftarrow \widehat{\sigma}(\mu_s^\tau, x_s)$ 
       $\widehat{\Delta}_s^T \leftarrow \widehat{\Delta}_s^\tau$ 
    endif
  return
endif
   $\tau \leftarrow \tau + 1$ 
   $U_\tau \leftarrow \{g_{s+\tau} \in G_O \wedge \text{executable}(\widehat{x}_{s+\tau-1})\}$ 
  ;; for each of the ground action candidates
  foreach ( $e_{s+\tau} \in U_\tau$ ) do
     $U_\tau \leftarrow U_\tau \setminus e_{s+\tau}$ 
    ( $\widehat{x}_{s+\tau}, \widehat{\Delta}_s^\tau$ )  $\leftarrow \text{execute}(e_{s+\tau}, \widehat{x}_{s+\tau-1})$ 
     $\mu_s^T \leftarrow \mu_s^\tau \cup \{e_{s+\tau}\}$ 
     $\widehat{\sigma}(\mu_s^T, x_s) \leftarrow \widehat{\sigma}(\mu_s^\tau, x_s) + \beta^\tau \sigma(e_{s+\tau}, \widehat{x}_{s+\tau-1})$ 
    ExhaustiveSearch( $\pi_s^T, \widehat{\sigma}(\pi_s^T, x_s), \widehat{\Delta}_s^T, \widehat{x}_{s+\tau}, \tau, \mu_s^T, \widehat{\sigma}(\mu_s^T, x_s), \widehat{\Delta}_s^T$ )
     $\mu_s^T \leftarrow \mu_s^T \setminus \{e_{s+\tau}\}$ 
     $\widehat{\sigma}(\mu_s^T, x_s) \leftarrow \widehat{\sigma}(\mu_s^\tau, x_s) - \beta^\tau \sigma(e_{s+\tau}, \widehat{x}_{s+\tau-1})$ 
  endfor
return ( $\pi_s^T, \widehat{\sigma}(\pi_s^T, x_s), \widehat{\Delta}_s^T$ )
endfunction

```

Figure 8: Search process for the optimal policy in Pseudo-code