Plan-Based Policy-Learning for Autonomous Feature Tracking

Maria Fox and **Derek Long** and **Daniele Magazzeni**
Department of Informatics
King's College London, WC2R 2LS, UK
email: *firstname.lastname@kcl.ac.uk*

Abstract

Mapping and tracking biological ocean features, such as harmful algal blooms, is an important problem in the environmental sciences. The problem exhibits a high degree of uncertainty, because of both the dynamic ocean context and the challenges of sensing. Plan-based policy learning has been shown to be a powerful technique for obtaining robust intelligent behaviour in the face of uncertainty. In this paper we apply this technique in simulation, to the problem of tracking the outer edge of 2D biological features, such as the surfaces of harmful algal blooms. We show that plan-based policy-learning leads to highly accurate tracking in simulation, even in situations where the uncertainty governing the shape of the patch cannot be directly modelled. We present simulation results that give confidence that the approach could work in practice. We are now collaborating with ocean scientists at MBARI to perform physical tests at sea.

1 Introduction

Plan-based policy learning has been successfully applied to multiple battery load management (Fox, Long, and Magazzeni 2011). The idea generalises very well to other problems where there is uncertainty that cannot be directly modelled, and against which control decisions have to be made under resource constraints. In recent discussions with ocean scientists and engineers at Monterey Bay Aquarium Research Institute (MBARI) we have been introduced to the problem of tracking the extent of submerged and partially submerged oceanographic features such as harmful algal blooms (HABs) (Das et al. 2010). Understanding the development, extent and decline of HABs is important in the ocean sciences because they are toxic to marine life and potentially to humans (Ryan et al. 2005). This paper describes the application, in simulation, of plan-based policy-learning to the problem of tracking the diffuse edge of the surface boundaries of such oceanographic features. We have focussed on following the surface boundaries of HABs, dealing with the dynamic nature of the bloom and the ocean environment. The physical situation is characterised by a high level of uncertainty, as the bloom disperses and changes at a rate determined by atmospheric and oceanographic conditions. Deciding how to detect the edge of the bloom and navigate its boundary with an unmanned vehicle, while conserving energy and avoiding danger, is a planning problem under uncertainty. Autonomous control of underwater vehicles in challenging missions is an area of great interest, with increasingly sophisticated approaches being deployed (McGann et al. 2008; Zhang et al. 2011).

The approach we have taken is based on sampling a large number of static surface bloom instances, which we call *patches*, solving each of these as a deterministic planning problem, and then learning a general purpose policy by classification methods. We make an important advance over the method used by Fox *et al.* (2011), which is to learn improvements to the policy during a second training phase. In this paper we explain our approach in detail, discuss its relationship with other approaches and present experimental results that demonstrate the effectiveness of the method, and its robustness to the shape of the surface patch, in simulation.

The next stage in our work is to test our method at sea, in collaboration with oceanographers at MBARI, and to extend our approach to tracking the 3D structure of submerged, and partially submerged, water features.

2 The Patch Tracking Problem in the Ocean

Harmful Algal Blooms develop in the coastal oceans due to oceanographic processes which cause biological particulate matter to come to the surface and bloom in the light (Glibert et al. 2005). Some of these organisms release chemicals into the ocean that are toxic to plant and marine life. Blooms are associated with large-scale marine mortalities and can lead to shell-fish poisoning events affecting humans.

The bloom-tracking problem, for our purposes, is the problem of following the edge of a surface patch with an Autonomous Underwater Vehicle (AUV) for a fixed amount of time, in order to help determine the shape and extent of the surface of the bloom. The AUV makes control decisions (what path to follow in order to remain close to the edge), based on taking regular readings with a chlorophyll sensor. The objective is to follow a contour within the patch that represents a particular chlorophyll concentration, allowing the surface of the bloom to be tracked and mapped over a series of missions. A chlorophyll threshold defining the contour to be followed is used to determine whether the AUV is inside or outside the contour at each time that a chlorophyll reading is taken. It is important to realise that the contour can-

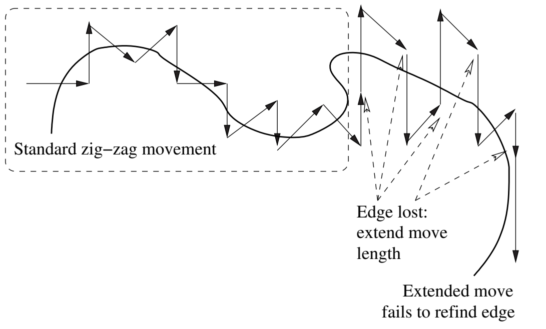
Standard zig−zag movement
Edge lost: extend move length
Extended move fails to refind edge

Figure 1: Possible traverse-widening control strategy in response to losing the target contour.

not be sharply defined because of the limitations of the sampling rate of the instrument and the interactions between the movements of the vehicle and of the water. Vehicle control is also constrained by the manoeuvrability limits, localisation accuracy and trajectory constraints (the vehicle yoyos along a sinusoidal path from the surface to about 15m depth or more, and back). The instrument can therefore only provide a reading for chlorophyll at points some metres apart, and this is insufficient to identify whether the vehicle is on the relevant contour — instead, it is only possible to determine whether the vehicle is *inside* or *outside* the area enclosed by the contour. Because of the degree of noise and local fluctuation in the chlorophyll readings, the AUV cannot be equipped with a pair of sensors mounted to simultaneously read values that span the contour, precluding the use of a control strategy based on the kind of simple edge-following that one can program into a line-following robot. Instead, to identify the approximate path of the contour requires that the AUV alternately read values that span the contour value, crossing the contour backward and forward to achieve this. The AUV must make a decision about which way to turn, in order to follow the contour, based on the combination of the most recent measurement and the history of observations. The path of the AUV should ideally zig-zag over the edge, turning into the bloom when a reading below the threshold is taken, and turning out of the bloom when the most recent reading is above the threshold. When a series of successive readings remain above or below the threshold, so that the contour has been lost, the AUV has to take corrective action to get back to the contour path. Knowing in which direction to turn to relocate the contour is the part of the problem that makes it complex enough to benefit from planning. The problem can be contrasted with edge- or line-following, where a single pair of readings straddling the edge or line indicates whether the vehicle is on the correct path or not, and the vehicle can detect a turn in the edge or line because one of its sensors reads an unexpected value.

At first sight the patch-tracking problem looks like a reactive control problem in which the AUV simply responds to the last reading by turning left or right. However, this reactive approach will not work when the AUV loses the edge of the patch, as there is then no guidance about which way to turn. The path followed by a reactive controller could be designed to increase the traverse distance until the edge is recovered (see Figure 1), but recovery will never happen if the edge has veered away at an angle (the traverse distance will just continue to increase and the AUV will remain lost). Alternative strategies include spiralling outward from the point at which the contour is lost, attempting to rediscover it by crossing it, but a significant risk is of rediscovering the contour while travelling in the wrong direction, leading to doubling back along the contour along a part that has already been explored. The difficulty in solving the contour-following problem depends on several factors: the distance the robot moves between decision points, the turning circle of the robot at those decision points and the relative curvature of the contour being followed.

Construction of a good strategy, providing a robust control choice in each situation, is the task of defining a *policy*: a function from the observed state (possibly including internal state such as memory of earlier observations and actions) to an action to execute next. Typical control strategies for robot line-following (F.Martin 1996) represent hand-built policies (often exploiting PID control) that solve (or partly solve) a similar problem. The challenge in the patch-edge following problem is that there is no obvious solution strategy to hand-code as a policy.

3 Policy learning for Patch Tracking

Experience with tracking similar shaped patches in the past can assist an AUV in deciding how to react in its current situation, including how best to react to losing the contour. This experience can be exploited by learning a policy from examples of good solutions to tracking behaviour. By learning a policy we provide the AUV with advice about which way to turn based on prior experience of having traversed a large number of surface patches in the past. The AUV will turn left (or right) if it has normally turned left (or right) in similar situations in the trajectory around previously seen blooms. As our experiments show, this learned policy behaviour often guides the AUV successfully back onto the edge of a patch when it has temporarily lost track of the appropriate contour on its zig-zag path. It turns out that learning this response from previous experience on a large number of similar patches leads to good behaviour at run-time, even on patches that are different in shape to those seen in training. Our results show a high level of robustness to shape and dimensions of surface patches.

The key to our approach is to exploit sampling (Metropolis and Ulam 1949) and classification to learn a powerful general policy. A benefit of sampling is that it is not necessary to have an explicit representation of the uncertainty in the problem domain. Instead, a large number of deterministic instances are chosen in such a way that they cover a broad variety of interesting situations likely to be seen in practice. Each of these instances can be independently solved using deterministic methods. Solving these generates the examples for the training phase of the approach. When all of the instances are solved, decision-tree learning can be applied to the problem of classifying combinations of decision variables with action choices. Each such combination of decision variables is called a *state*, and the association of an action with each state results in a collection of state-action pairs which form a *policy*. We use the J48 classifier (provided by the WEKA toolkit (Hall et al. 2009)), and the resulting policy is in the form of a decision tree. Depending on the number of samples used in training, the method of policy-learning takes several days to produce a policy which then exhibits robust, general behaviour.

Our approach finds a compact representation of the best thing to do next in any patch-tracking situation. The approach is based on sampling, rather than reinforcement learning, because sampling provides a scalable alternative to explicit modelling of the uncertainty in the domain. It extends the work of Fox *et al.* (2011) by applying the sampling strategy not only for learning an initial policy, but also in subsequently improving its quality, making it able to recover after bad decisions.

The policy-learning approach is described in detail in section 4. Figure 2 illustrates the approach schematically. Starting with a large number of static instances of patches, we train a classification-based learning method by solving each case independently using deterministic planning. The solutions to these cases are then fed in as input to a decision tree classifier that produces, using J48 decision tree learning, a condensed representation of the training history as a policy. The policy can then be used on-board the AUV to decide the best thing to do next in any situation that arises during a patch-tracking mission. Figure 2 shows, above the dotted line, the policy learning approach described by Fox *et al.* (2011). Below the line is the second learning phase that the current work contributes to the policy-learning approach. The learned policy can be seen as a highly intelligent controller that condenses the vast experience recorded in its state variables, selecting the next action without search.

Related approaches to policy construction have been explored by other researchers (Yoon, Fern, and Givan 2002; Teichteil-Königsbuch, Kuter, and Infantes 2010; Sanner and Boutilier 2009). However, MDP-based approaches cannot be applied in this domain as there is not a probabilistic model to describe the uncertainty involved in the problem, and then it is not possible to define the transition function of the MDP.

The approach most closely related to ours is Hindsight Optimisation (HO) (Chang, Givan and Chong 2000; Fern, Yoon, and Givan 2006; Eyerich, Keller and Helmert 2010) which has become a well-researched technique for learning policies based on plans. The HO technique works by sampling a large number of deterministic instances of an MDP with initial state s, then solving these instances using a deterministic planner over a fixed horizon. Finally, the estimated value for the state s is computed as the average value obtained from the deterministic plans. A potential problem is that the plan states are abstracted from reality and might not match the situations encountered at execution time. The key to our approach is to distinguish between *plan states* and *policy states*. Policy states comprise observable variables which are correlated with the decision variables of the plan states. This allows observations to be identified with the corresponding plan states so that the appropriate actions are selected at policy execution time. We call our approach *observable correlate* policy-learning to emphasise this feature.

As is always the case with sampling approaches, the samples do not cover all possible cases that might be encountered in practice. It is therefore necessary to have a *default* action that can be applied when the policy cannot offer a sensible action choice. Our plan-based approach attempts to minimise the number of times that a default action must be taken. To design good default actions, we experimented with providing a bounding box around the patch being tracked, with special actions designed to navigate the AUV from the edge of the box back onto the patch, in cases where the policy could not relocate the patch before hitting the edge of the bounding box. In the last phase of the learning stage, we run a set of patch-tracking missions on unseen patches in order to find weaknesses in the policy learned so far. On each mission we store all the default actions that are executed on the trajectory round the patch, resulting in a new set of small sequences of state action pairs. We then add this new set to the original training data and we relearn a new policy from this extended training set. This represents an important extension of the approach presented by Fox *et al.* (2011), enhancing the robustness and applicability of the policy, as described in Section 4.7.

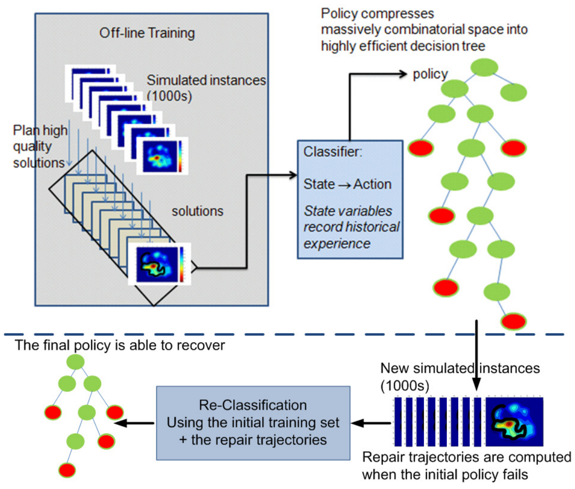
Off-line Training
Simulated instances (1000s)
Plan high quality solutions
solutions
Classifier:
State → Action
State variables record historical experience
Policy compresses massively combinatorial space into highly efficient decision tree
policy
The final policy is able to recover
Re-Classification Using the initial training set + the repair trajectories
New simulated instances (1000s)
Repair trajectories are computed when the initial policy fails

Figure 2: The plan-based policy-learning method

4 Patch Tracking as a Planning Problem

In this section we explain how we solve the patch-following sample problems, beginning with a consideration of why these problems can usefully be seen as planning problems.

4.1 Why Planning?

Each patch sample presents an edge-tracking problem that must be solved as a deterministic instance. In each instance, the AUV can use a small set of actions (forward moves and turns) and these must be assembled into a sequence that starts from the initial AUV position and tracks the edge.

Once a patch sample is created, the precise structure of the edge of the patch for that sample is known. This means that a simple edge-following algorithm could be used to follow the edge without any deviation and would appear much more appropriate to the task than a general purpose planner.

However, such a solution highlights one of the potential pitfalls of using a sampling approach to manage uncertainty: in the generation of a sample the uncertainty in the original problem is removed, but the problem we must ultimately solve will retain the uncertainty. For the training samples to convey useful information to help in solving the real, uncertain, instances, there must be some way to connect the observable state available to the AUV in the real instances to the action selections made in the training samples.

To ensure an adequate training set it is necessary to generate solutions that visit a wider range of states than are required to simply track the patch edge, and that make action selections for reasons other than that the selected action immediately crosses the patch boundary. We achieve this by setting a *confusion* limit in the action preconditions which forces the planner to cross the edge every n moves (where n is the bound). This leads the planner to compromise between the length of the path and the need to follow the patch edge, making the problem an interesting and difficult optimisation task. It is this requirement (discussed further in Section 4.2) that makes the problem one for which planning is a suitable approach, with planners offering a route to rapid and flexible solution.

4.2 Deterministic Problem Modelling

Given the bounding box enclosing the patch, we model it as a grid of $k \times k$ cells. Each cell is labelled either black or white, indicating whether the cell is inside the patch or outside, respectively. The patch is then described as a PDDL problem by giving the set of all the connections between adjacent cells, together with the colour of each cell. A fragment of a PDDL problem is shown in Figure 3.

```
(define (problem patchtracking1)
 (:domain patchtracking)
 (:objects
   bl wh - colour
   dummy_N, dummy_E, dummy_S, dummy_W,
   x17_58, x17_59, x18_55, x18_56
   [...]   - location
   auv     - vehicle
   n s e w nw ne sw se - direction)

 (:init
  (at auv x18_59)
  (facing auv n)
  (= (confusion auv) 2)
  (linked x17_58 x17_59 s)
  (linked x17_58 x18_59 se)
  (linked x17_58 x18_57 ne)
  (white x17_58)
  [...]
))
```

Figure 3: Part of PDDL problem for patch tracking

The action the AUV can perform is to move between two connected cells. We restrict possible moves to the following directions, to reflect constraints on manoeuvrability of the AUV: left, forward-left, forward, forward-right, right. Furthermore, in the PDDL domain we distinguish a move between cells of different colours from a move between cells of the same colour. The former is used to cross the edge of the patch, while the latter is used when the AUV remains inside or outside the patch. The AUV has a *confusion* level which measures how far it has travelled since last crossing the edge: the confusion level increases when the AUV moves through cells of the same colour, and drops to zero when the AUV crosses the edge, as shown in Figure 4.

```
(:action move
 :parameters
  (?v - vehicle ?from ?to - location
   ?dold ?dint ?dnew - direction
   ?c1 ?c2 - colour)
 :precondition
  (and (< (confusion ?v) 4) (new ?to)
       (colourOf ?from ?c1) (colourOf ?to ?c2)
       (eq ?c1 ?c2) (at ?v ?from)
       (facing ?v ?dold)
       (linked ?from ?to ?dint)
       (validDirection ?dold ?dint ?dnew))

 :effect
  (and (increase (confusion ?v) 1)
       (at ?v ?to) (not (at ?v ?from))
       (facing ?v ?dnew)
       (not (facing ?v ?dold)) (not (new ?to))))
```

Figure 4: Part of the PDDL domain for patch tracking

Since the patch is known, the planner can plan to keep the confusion level below a given threshold by choosing to cross the edge as often as necessary while keeping the plan as short as possible. For example, given a threshold of 4, the plan has to zig-zag across the edge with one crossing at least every 4 moves. The benefit of this is that the policy-learning process will be exposed to situations where the AUV has encountered a high confusion level, and it will learn the best thing to do in such cases.

We use LPRPG (Coles et al. 2008), a heuristic forward search planner, to solve the problems. However, the problem we are interested in presents a huge state space, as the patches to be tracked cover an area of several kilometers squared, resulting in grids of 180×180 cells. To make these problems accessible we decompose them into a set of subproblems, as described in the following section.

4.3 Planning Problem Decomposition

The patch is divided into 9 squares, as shown in Figure 5 (a), and we then track the patch edge in each square. The planner therefore deals with smaller grids and generates a set of partial solutions corresponding to sub-paths.

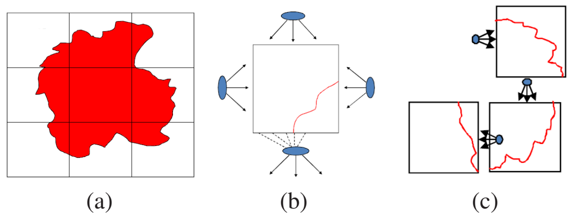

Figure 5: Decomposing and re-linking partial solutions

However, since each square represents a different planning problem, it is necessary to build connections between different partial solutions. Dummy nodes are used, as shown in Figure 5 (b), to specify initial states and goals. All the cells in the grid boundaries are connected to the nearest dummy node. In each subproblem, the goal is to traverse between a dummy node A and a dummy node B while keeping the confusion level below the threshold. LPRPG can solve each subproblem (a 60×60 grid) in less than 10 seconds.

Finally, the partial solutions are linked to each other to obtain the complete path, as sketched in Figure 5 (c).

4.4 Policy Learning

Having shown how to generate plans for a known patch, we now focus on how to deal with unknown patches. In general, it is not possible to predict the exact shape of the surface patch of a bloom. However, satellite imagery can be used to approximate its extent in order to define the bounding box enclosing the patch. We also have a probabilistic model, implemented as a Matlab patch generator[1], which characterises typical chlorophyll distribution and that can be used to generate a representative set of cases. An example of a patch is shown in Figure 6, where different contours are highlighted according to chlorophyll thresholds.

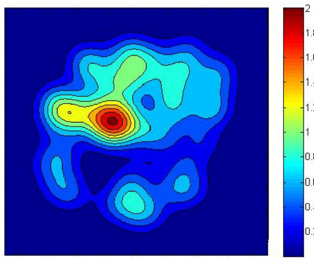

Figure 6: Example of chlorophyll distribution in a patch

In order to learn a policy, the patch generator is used to create a set of deterministic problems which can be solved using the approach described above. All of the patches generated for learning were roughly square or circular, approximately filling the bounding box region and having no internal structure. We refer to this kind of patch as *standard*. The generated plans can then be aggregated through classification to obtain a policy.

4.5 Policy State

The key to observable correlate policy learning lies in selecting a suitable policy state. In fact, an informative set of state variables is essential to enable the classifier to structure the decision tree so that particular actions are preferred over others in particular states.

The states used as the basis of classification cannot be the same states as are used for planning, since only observable state values can be used to apply the policy in execution. Identifying appropriate state variables is the hard problem of feature selection encountered in all machine learning tasks (Liu and Yu 2005). After some experimentation, we adopted the following state as the basis for classification:

$$s = (\theta, L_c, R_c, F_c, FL_c, FR_c, P, F, C)$$

[1]Provided by Michael Godin, Software Engineer, MBARI.

where:

- $\theta \in [0, 360)$ is the average bearing over the last 10 moves;
- $L_c, R_c, F_c, FL_c, FR_c \in \mathbf{N}$ count the number of times each of the five actions (Left, Right, Forward, Forward-Left, Forward-Right, respectively), has been performed in the plan so far;
- $P \in \mathbf{B}$ is true if the last visited cell was within the patch, false otherwise;
- $F \in [N, S, E, W]$ denotes the current facing direction;
- $C \in \mathbf{N}$ is the confusion level.

The values of the policy state variables are constructed by simulated execution of the plans generated to solve the sample set. Each observation generated in a trace will be associated with the corresponding action choice made in the plan, in order to build the entire training data set. The correlation between observables and the action choices is then determined by the classification process.

The policy maps values of these states to actions and, during execution, will give rise to a sequence of transitions of the following form (assuming, for example, that the action $a = \textit{Go Right}$ is performed):

$$s(t) = (\theta, L_c, R_c, F_c, FL_c, FR_c, P, F, C) \xrightarrow{a}$$
$$s(t') = (\theta', L_c, R_c + 1, F_c, FL_c, FR_c, P', F', C')$$

where θ' and F' are the new average bearing and the new facing direction, respectively, P' is true if the last visited cell is within the patch, otherwise is false, and $C' = C + 1$ if $P' = P$, else $C' = 0$.

4.6 Classification

Following Fox *et al.* (2011), classification is performed using WEKA (Hall et al. 2009). In particular, we used the `J48` classifier, which implements the machine learning algorithm C4.5 (Quinlan 1993). The output is a decision tree whose leaves represent, in this context, the next AUV action.

We found that the best results are obtained by classifying $2,000$ plans, as further extending the training set does not make any significant improvement to the policy performance, but increases memory and time requirements. As each plan consists, on average, of 500 actions, the classification process involves considering about 10^7 values.

We converted the resulting decision tree into a C program, in the form of a set of nested if-then-else statements checking the values of the state variables and returning the corresponding action. The compiled policy is about 2MB in size and takes negligible time to propose each action.

4.7 Recovery Action

It is not possible for the policy to cover the whole space of reachable states since it is too large. Therefore the policy must be supplemented with a sensible default action to be used when the AUV gets lost at execution time. This happens when the input state is too distant from any of the states encountered during learning. In our setting, a default action is required when the AUV confusion level exceeds the critical level used in planning. Just as the selection of the variables for the policy state is key to performance of the policy, so the choice of the recovery action is key for determining the robustness of the policy.

We first tried ‘bouncing’ the AUV off the bounding box that encloses the patch. Thus the default action was to continue going ahead until the bounding box edge was reached, and then be deflected back onto a path that would cross the patch edge and allow the AUV to relocate it. However, this solution presents two problems: first, it is not efficient if the bounding box is large and, second, the use of this default action impacts on the rest of the policy execution because bouncing off the box dramatically distorts the value of the counter variables. Using the policy after this default action results in poor behaviour. This is clearly shown in Figure 7, where the ideal and actual behaviour of this solution are compared.

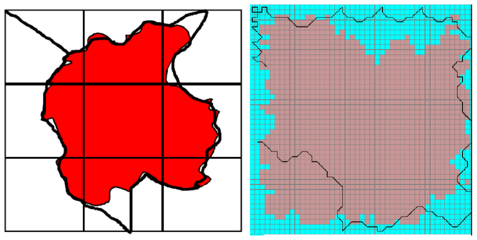
Figure 7: Ideal (left) and actual (right) behaviour of the *bouncing box* repair action

To overcome this problem, we moved an area-based repair action. Figure 8 (left) shows the direction selected in each of the sub-areas, according to whether the AUV was last inside or outside the patch. The repair action moves the AUV in a straight path following the indicated bearing.

This solution performs well: an example of its behaviour is shown in Figure 8 (right). In this test, repair was triggered when the confusion level increased beyond a threshold of 5. Interestingly, the area-based default action results in a state consistent with the policy, which is still able to correctly handle the AUV states obtained after default actions have been applied.

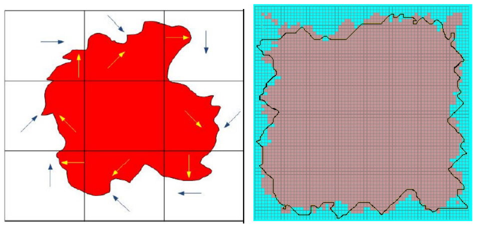
Figure 8: *Area-based* repair action and its effects

4.8 Policy Improvement through Policy Rollout and Re-Classification

We perform a second training phase in which we improve the policy by presenting it with situations that require a default action after the first training phase. We ran a set of 10,000 simulated tests, setting a confusion level threshold of 2. Lowering the confusion level threshold compared with the original planning model allows us to increase the policy sensitivity. During each test, we stored all the area-based actions triggered when the policy could not match the input state. This resulted in a set of small sequences of state action pairs which we added to the training set used for learning the initial policy. We then relearned a new policy from the extended training set. All of this work is done offline before the policy is applied to “real” patches.

The improvement made by this phase is twofold: first, the new policy is more sensitive than the initial one, as it tends to keep the confusion level below the very small threshold of 2. Second, it is more robust, as the policy rarely fails to identify the state. In particular, the policy now handles states in which the AUV confusion level is greater than the critical threshold that would normally trigger a repair action.

5 Evaluation

In this section we discuss the simulation framework that we used to evaluate our approach, and the comparisons we performed with a non-planning-based strategy. In our experiments, we assume to know the coordinates of the bounding box enclosing the patch, and that the AUV can be commanded to navigate to a waypoint on the North-West corner of the patch before deploying the policy. These assumptions are consistent with the setting used in real sea tests.

5.1 Static Policy

In order to evaluate the advantages of using a plan-based policy we compare it with a hand-coded policy. We developed a *static policy*, based on the simple strategy of simply using the default actions described in Section 4.7. This is equivalent to using the plan-based policy with a confusion level threshold of zero: in this case the learned policy is never applied triggered and the default action is used in every state.

A static policy using the bounding box default actions (see Figure 7) proved unable to follow the edge of the patch. The static policy using the area-based default actions is reasonably effective, corresponding closely to hand-coded line-following policies. We enhanced this static policy with a *loops-avoidance* routine, to handle positions of the AUV where the area-based actions fail to escape a local structure (eg the boundary of an internal structure within a patch).

This policy performed very well on the patches used in our training set: an example is shown in Figure 9 (left). However, a deeper evaluation, described below, shows that this static policy lacks robustness, which is a crucial requirement for the patch-tracking application.

In order to evaluate the effectiveness of the policies, we ran a large set of simulations, using the Matlab patch generator. We first considered a chlorophyll threshold of 0.2, that results in roughly circular patches (such as Figure 6). These patches are similar to the cases we used as training examples for learning the policy. The policy performed well, as might be expected, since the patches are similar to the training data. An example is shown in Figure 9 (right).

5.2 Policy Efficiency Measures

The purpose of the patch-following behaviour is to attempt to identify the contour of the patch as efficiently as possible.

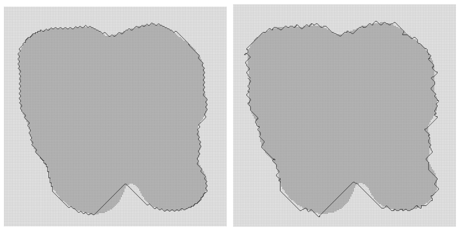

Figure 9: Standard patch with a chlorophyll threshold of 0.2 tracked using static (left) and plan-based (right) policy

Because the AUV will collect data throughout its path, it will be possible to reconstruct from the data a view of which parts of the trajectory are close to the contour. Therefore, provided the AUV approaches close to all parts of the contour, the mission will successfully trace the patch, although possibly inefficiently. If the AUV only follows a path that is close to the contour then it will be efficient, although it might not visit all parts of the contour. We therefore consider two measures in determining how successfully policies perform, letting $\mathcal{E}$ be the set of points along the edge of the patch and $\mathcal{T}$ be the points along the AUV trajectory:

- *Minimal distance of the patch edge from the trajectory.*

$$\Psi = \frac{\sum_{\forall e \in \mathcal{E} \cdot t^* \in \mathcal{T} \cdot \nexists t' \in \mathcal{T} \cdot ||e-t'|| < ||e-t^*||} (||e - t^*||^2)}{|\mathcal{E}|}$$

  Ψ measures how closely the AUV tracks the patch-edge.

- *Minimal distance of the trajectory from the patch.*

$$\Omega = \frac{\sum_{\forall t \in \mathcal{T} \cdot e^* \in \mathcal{E} \cdot \nexists e' \in \mathcal{E} \cdot ||t-e'|| < ||t-e^*||} (||t - e^*||^2)}{|\mathcal{T}|}$$

  Ω measures how efficiently the AUV tracks the patch.

Both of these measures sum squared distances, similarly to a sum-of-squares error measurement for curve fitting.

We also measure the average *confusion level* of the AUV. This reflects the global performance of the AUV in tracking the patch edge, showing approximately how close to the contour the AUV passes throughout its path.

5.3 Policy Robustness

Figure 6 indicates that the shape of the contours vary significantly at different chlorophyll levels. In fact, the variation in contours in that Figure is also reflected between the outer shapes of HABs in practice, due to effects of ocean currents, wind and distribution of nutrients. The patches used for training are all roughly circular, but the shapes met in practice vary between long and narrow patches to multiple closely linked islands of high chlorophyll. In order to be useful, the AUV control policy must adapt well to these different situations. Therefore, together with the standard (roughly circular) patches, we also considered three other kinds of patches: *horizontal*, *thin* and *inner-structured* patches, obtained using the patch generator with different parameters (see Figures 10-12).

Bloom	Policy	Ψ	Ω	Conf	Length
Standard	static	39.37	2.56	2.52	510.94
	planning	39.48	3.28	3.59	533.10
Horizontal	static	1741.18	51.72	19.93	328.77
	planning	187.83	15.08	14.67	387.23
Thin	static	56.95	45.14	45.14	314.39
	planning	25.09	35.80	9.71	353.60
Inner	static	1009.51	21.40	16.23	358.45
	planning	585.66	20.07	15.56	382.94

Table 1: Policy performance and robustness evaluation

Horizontal patches are flatter than they are wide, representing just a segment of the kind of patch used in training. Thin patches are very flat and wide, representing thinner such segments. Inner-structured patches are more interesting: they are broken up into several loosely connected or disconnected regions, which are close enough together to be considered part of the same patch.

We only use standard patches for learning the policy as mixing different shapes leads to a poorer decomposition of the decision tree and weaker basis for classification. Furthermore, although the three shapes above represent some typical patches, it is not possible to anticipate all the variability, and during a mission the system cannot know what type of patch is on, so it could not select an appropriate policy from a library. Thus, the policy has to be robust enough to cope adequately with whatever it finds.

6 Results

The efficiency measures defined in Section 5.2 were used to evaluate the performance of the static and plan-based policies against standard, horizontal, thin and inner-structured patches. We used 10,000 samples of each type.

Table 1 shows the results of the comparison. For all the measures, the lower the value (which refers to the average value over 10,000 tests for each kind of patch), the better. As can be seen, the plan-based policy significantly outperforms the static one, proving a high degree of robustness. The two policies perform quite similarly when dealing with standard patches, which is to be expected, because they cover the area inside the bounding box quite evenly and with a convex edge, making them straightforward for both policies.

In contrast, horizontal, thin, and inner-structured blooms are the most challenging as their shapes are quite irregular. With respect to them, the plan-based policy performs surprisingly well, reducing the Ψ values by up to one order of magnitude compared with the static policy, when dealing with horizontal patches and providing a very significant improvement in all the other cases.

Figures 10-12 show some representative examples of the behaviour of the policies for each kind of patch and give an indication of how much more robust is the plan-based policy.

When dealing with *horizontal* patches (Figure 10), the static policy starts zig-zagging over the edge of the patch, but then gets lost inside it. In contrast, the plan-based policy successfully follows the edge of the patch. The fact that the static policy does not guide the AUV around the horizontal blooms, while the plan-based one does, is confirmed more generally by the big difference ($\simeq 90\%$) in the Ψ values in

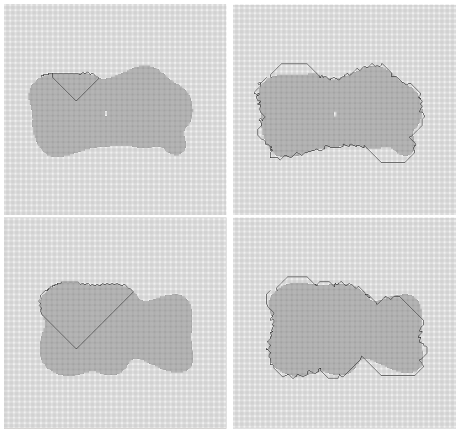

Figure 10: *Horizontal* blooms tracked using static (left) and plan-based (right) policy

Table 1. Note that, despite the small flaws in the upper left and lower right corners of the patch, where the AUV confusion level increases, the plan-based policy is able to get the AUV back to the edge very quickly and providing a clear and almost perfect picture of the extent of the patch.

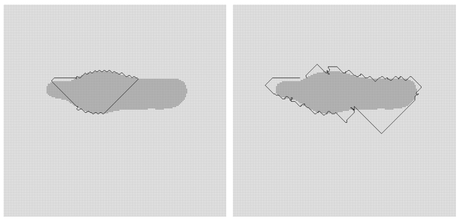

Figure 11: *Thin* bloom tracked using static (left) and plan-based (right) policy

A similar behaviour is observed with *thin* blooms (Figure 11), where the Ψ values differ by $\simeq 50\%$ and the confusion level is dramatically reduced by the plan-based policy.

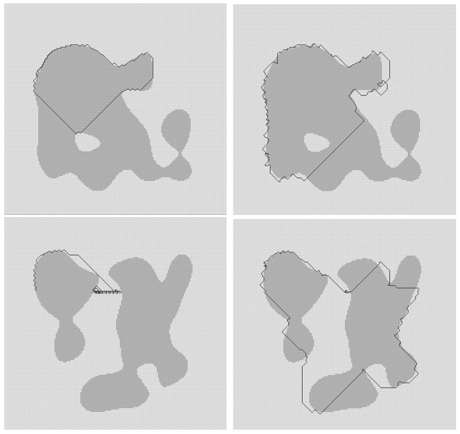

Figure 12: *Inner* blooms tracked using static (left) and plan-based (right) policy

Finally, *inner-structured* patches (Figure 12) are obtained in the simulator by considering a chlorophyll threshold of 0.5. This gives rise to very irregular shaped patches, which are very difficult to handle. Indeed, as can be seen in Figure 12 (top), in this case the static policy gets lost inside the patch. This also happens to some degree with the plan-based policy, but it does provide a better view of the extent of the patch. It is very common to find inner-structured patches split into separated partial patches. An example can be found in Figure 12 (bottom) that also shows a recurrent behaviour of the static policy when dealing with split patches. Once the AUV leaves the first patch, the static policy is not able to direct the AUV towards the new patch, and the AUV simply follows a holding pattern.

On the other hand, the plan-based policy drives the AUV towards the second patch, then it follows a part of its boundary and finally gets back to the first patch, providing quite a good picture of the union of the two patches. Note that, although the path followed by the AUV appears to represent a single patch, a quick look at the log file with data recorded along the trajectory would show two set of points where the chlorophyll readings fell and confusion level increased, corresponding to the path fragments where the AUV was traversing the areas between the two sub-patches.

7 Conclusion

In this paper we have described an application of plan-based policy learning to the problem of autonomously tracking the boundary of the surface of a partially submerged harmful algal bloom. The approach could also be applied to guiding an AUV attempting to follow other water features such as temperature gradients. Furthermore, blooms are actually three-dimensional masses suspended in the ocean, although only the surface is visible from above. Oceanographers are interested in studying the extent of the submerged part of the bloom as well as its surface patch. The approach we have described can be generalised to the 3D case, but is complicated by the fact the AUV must follow a trajectory that yoyos through the water column. This is the subject of future work.

The approach described here builds on the work of Fox *et al.* (2011) on using plan-based policies for efficient multiple battery load management. We have demonstrated that the approach generalises to a very different situation, characterised by very different properties and constraints. In this work we aim to closely follow a diffuse contour. We show that plan-based policies perform very well on surface patch-tracking problems and are robust to different patch shapes. Our experiments show that our plan-based approach outperforms a static policy that exploits an intelligent default action but is not based on planning. We have begun a collaboration with oceanographers to test our approach at sea.

Acknowledgments

This work was partially funded by the EU FP7 Project 288273 PANDORA and the EPSRC Project “Automated Modelling and Reformulation in Planning” (EP/G0233650). We would like to thank Kanna Rajan and Frederic Py for their explanation of Harmful Algal Blooms and their insights into the patch-tracking problem.

References

Chang, H. S.; Givan, R.; and Chong E. K, P. 2000. On-line Scheduling via Sampling. In *Proc. of Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 62–71.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. A hybrid relaxed planning graphlp heuristic for numeric planning domains. In *Proc. of Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 52–59.

Das, J.; Rajan, K.; Frolov, S.; Py, F.; Ryan, J.; Caron, D. A.; and Sukhatme, G. S. 2010. Towards marine bloom trajectory prediction for AUV mission planning. In *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, 4784–4790.

Eyerich, P.; Keller, T.; and Helmert, M. 2010. High-Quality Policies for the Canadian Traveler's Problem. In *Proc. of AAAI Conference on Artificial Intelligence*.

Fern, A.; Yoon, S. W.; and Givan, R. 2006. Approximate Policy Iteration with a Policy Language Bias: Solving Relational Markov Decision Processes. *J. Artif. Intell. Res. (JAIR)* 25:75–118.

F.Martin. 1996. Kids Learning Engineering Science Using LEGO and the Programmable Brick. In *Proc. of the Annual Meeting of the American Educational Research Association*.

Fox, M.; Long, D.; and Magazzeni, D. 2011. Automatic Construction of Efficient Multiple Battery Usage Policies. In *Proc. of Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 74–81.

Glibert, P. M.; Anderson, D. M.; Gentien, P.; Granéli, E.; and Sellner, K. G. 2005. The Global Complex Phenomena of Harmful Algal Blooms. *Oceanography* 18(2):130–141.

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. H. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explorations* 11(1).

Liu, H., and Yu, L. 2005. Toward Integrating Feature Selection Algorithms for Classification and Clustering. *IEEE Trans. on Knowledge and Data Engineering* 17(4):491–502.

McGann, C.; Py, F.; Rajan, K.; Ryan, J. P.; and Henthorn, R. 2008. Adaptive Control for Autonomous Underwater Vehicles. In *Proc. of 23rd AAAI Conference on Artificial Intelligence, (AAAI)*, 1319–1324.

Metropolis, N., and Ulam, S. 1949. The Monte Carlo Method. *J. Amer. Stat. Assoc.* 173(5-6):335–341.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Ryan, J. P.; Dierssen, H. M.; Kudela, R. M.; Scholin, C. A.; Johnson, K. S.; Sullivan, J. M.; Fischer, A. M.; Rienecker, E. V.; McEnaney, P. R.; and Chavez, F. P. 2005. Coastal ocean physics and red tides: an example from Monterey Bay, California. *Oceanography* 18:246–255.

Sanner, S., and Boutilier, C. 2009. Practical solution techniques for first-order MDPs. *Artif. Intell.* 173(5-6):748–788.

Teichteil-Königsbuch, F.; Kuter, U.; and Infantes, G. 2010. Incremental plan aggregation for generating policies in MDPs. In *Proc. 9th Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*.

Yoon, S. W.; Fern, A.; and Givan, R. 2002. Inductive Policy Selection for First-Order MDPs. In *Proc. of Conf. on Uncertainty in AI (UAI)*, 568–576.

Zhang, Y.; McEwen, R. S.; Ryan, J. P.; Bellingham, J. G.; Thomas, H.; Thompson, C. H.; and Rienecker, E. 2011. A peak-capture algorithm used on an autonomous underwater vehicle in the 2010 Gulf of Mexico oil spill response scientific survey. *J. Field Robotics* 28(4):484–496.