2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

AUV Mission Control via Temporal Planning

Michael Cashmore*, Maria Fox*, Tom Larkworthy†, Derek Long*, and Daniele Magazzeni*
*King's College London, UK
firstname.lastname@kcl.ac.uk
†Heriot-Watt University, Edinburgh, UK
firstname.lastname@gmail.com

***Abstract*—Underwater installations require regular inspection and maintenance. We are exploring the idea of performing these tasks using an autonomous underwater vehicle, achieving persistent autonomous behaviour in order to avoid the need for frequent human intervention. In this paper we consider one aspect of this problem, which is the construction of a suitable plan for a single inspection tour. In particular we generate a temporal plan that optimises the time taken to complete the inspection mission. We report on physical trials with the system at the Diver and ROV driver Training Center in Fort William, Scotland, discussing some of the lessons learned.**

I. Introduction

We are exploring the problem of autonomous inspection and maintenance of a seabed facility. The autonomous inspection and maintenance is to be carried out using AUVs over extended horizons without human intervention. This demands sophisticated controllers to manage the necessary navigation and interaction tasks, as well as planning, to assemble the lower level tasks into a coherent mission plan over a rolling horizon. This paper explores the integration of these two aspects, focussing on the modelling of a temporal planning domain for planning inspection missions around seabed installations, managing the possibility that the structure we are inspecting is different in some way from our expectations, requiring closer inspection of particular parts or of different parts to those originally planned. Importantly, the model must be simple enough to allow the planner to manage the high-level reasoning, leaving the execution of navigation and interaction to the controller, while at the same time maintaining a level of consistency with the physical environment that ensures plan validity. We have implemented a system that integrates planning and execution in the Robot Operating System (ROS) framework [1]. A preliminary version of the framework has been tested in simulation and is described in [2]. In this paper we present a significant extension of that work, and we describe a new planning formulation of the problem which takes into account the temporal features of the problem and we report on physical trials performed in a large underwater tank at Fort William, in Scotland.

The use of AI Planning in AUV mission control is not new. Recent works include [3][4] (where the constraint-based temporal planning system EUROPA-2 is used in the T-REX framework), [5][6] (where homotopy classes are used to guide the path planning), [7] (where a coarse plan is generated to initialise the inspection of an unknown hull), [8] (where sampling-based motion planning is used to solve missions modelled in LTL), or [9] (where a plan-based policy is used to guide an AUV for autonomously tracking the boundary of the surface of a partially submerged harmful algal bloom). However, with respect to the state-of-the-art in planning AUV missions, in this paper we describe the following new contributions:

- we present a new approach for modelling the environment for the AUV inspection task, based on using a probabilistic roadmap [10], [11] (PRM) extended with a set of strategic points;
- we present a *temporal* planning domain for the AUV inspection task, which is based on an abstraction of angles and orientations which are represented symbolically, and we show how a temporal planner can be used to find efficient plans;

These new features allow a more efficient use of state-of-the-art planners in planning AUV inspection tasks and an easier integration between the planner and the controller. This was confirmed by the successful physical trials that we performed at Fort William, and that we describe in the paper, presenting the results and discussing some of the lessons learned.

The paper is structured as follows. We begin with a discussion of the issues related to the integration of planning and control. In Section III we describe the process of planning inspection tasks, including the temporal model we designed. In Section IV we show how this process is integrated with the control layer for execution. In Section V we introduce and discuss the physical trials. Section VI concludes the paper.

II. Integrating Task Planning and Control

Planning is an Artificial Intelligence technology that seeks to select and organise activities in order to achieve specific goals [12]. A planner uses a domain model, describing the actions through their pre- and post-conditions, and an initial state together with a goal condition. It then searches for a trajectory through the induced state space, starting at the initial state and ending in a state satisfying the goal condition. In richer models the induced state space can be given a formal semantics as a timed hybrid automaton, which means that a plan can synchronise activity between controlled devices and external events.

When a plan is executed it is common to dispatch the actions in the plan by invoking a controller responsible

978-1-4799-3684-7/14/$31.00 ©2014 Crown

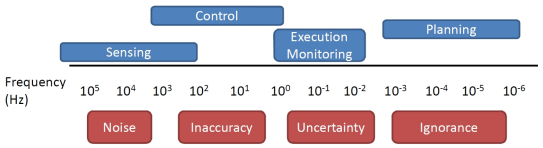


Fig. 1. Planning and control integration

for the realisation of each action as it occurs in the plan. The actions selected by the planner determine which controllers are invoked and when, but also supply parameters that inform the controllers of the specific requirements of each particular execution task. Planning is a high level reasoning activity coordinating the activities of the fundamental control and sensing components. Control, in turn, sits above sensing. Figure 1 shows these components and indicates a sense of the frequency of events with which they are typically associated. So, planning is a coarser-grained activity than the control systems its actions cause to be deployed. Control systems help to manage disturbances in behaviours during execution. These can be characterised as forms of uncertainty and different types of uncertainty also manifest at different frequencies: noise and inaccuracy (measurement accuracy and control inaccuracy) appear at the highest frequencies, while lower level frequencies capture uncertainty about action outcomes, identification of objects and the relations between them and, at lower frequencies still, there is the impact of ignorance which leads to the need to restructure and modify the world models available to reasoning systems. In this work we use replanning to tackle the effects of ignorance and uncertainty, while relying on robust control and signal processing systems to handle innacuracy and noise.

Because planning happens at a low frequency of events, much of the uncertainty encountered at execution is handled at these lower levels. For example, the control system handles the disturbance caused by unexpected currents, and sensor data interpretation handles noisy sensors. In the case where a controller fails to complete its action, the planner adapts to unexpected features or events by revising its abstract, deterministic model of the world and replanning to take account of the new features. For example, if strong currents preclude hovering at inspection points, rather than continue trying and failing to complete the doomed action, the planner would revise its approach, and begin dispatching a new plan, moving to new inspection locations.

A. Modelling Planning Domains for Execution

One of the design principles guiding development of domain-independent planning systems is to create stand-alone tools that might be used in a tool chain, taking input in a standard form (a domain file and a problem file, written in PDDL [13]) and generating a plan in a standard form. Exploiting a tool following this design requires an integration in which the necessary inputs are constructed and then that the output can be appropriately interpreted. Firstly, therefore, it is necessary to determine what actions are available to the system and to decide which of these actions can be exposed as choices to the task planner in the PDDL domain description. It is important to realise that not all of the actions available to the system itself need be exposed and there can be good reasons to avoid doing so. In particular, where actions are *forced* by the context, it is wasteful to require the planner to discover this by search. Construction of the domain model is usually only required once and can be performed by the system designer.

To construct the second input, the problem description, the system state must be interpreted and encoded in PDDL. This generally involves some kind of abstraction process, translating sensor data into symbolic representations. In addition, because the planner manipulates symbolic representations of objects and their relationships to the environment, while the controllers realising the planned actions usually require metric valued parameters, it is often useful to store data associated with the interpretation of objects and their relationships in preparation for the interpretation of the subsequent plan. For example, the coordinates of a waypoint are not important to the planner, but will be important in executing an action that requires a robot to visit the waypoint, so must be recorded as data associated with the waypoint name. The abstraction process affects not only the creation of the initial state, but also the creation of the goal of the planning problem. This is because goals that require the achievement of some state will require that the properties of the state be abstracted into the symbolic representation in the planner. Then, any plan that achieves the symbolic state will achieve only *some* state whose abstracted description satisfies the symbolic goal (assuming the translations, planning and plan execution all function flawlessly). In our context, the main task the planner must achieve is to observe structures. Observation as a symbolic task merely involves executing the *observe* action from an appropriate waypoint and targeting the intented inspection point. This is an abstraction of a process that is, in fact, significantly affected by sensor noise, environmental conditions and so on. As a consequence, it is insufficient to characterise the goal simply as to have executed a single observation of each inspection point. Instead, we refine the detail by allowing each observation to add to an accumulating score for the observations associated with an inspection point, according to the expected benefits of that observation based on the relative position of the waypoint and expected environmental features. The goal is then expressed as a requirement to accumulate sufficient observation value for each inspection point, which can be achieved by applying different alternative subsets of inspections, trading their number and position against expected benefits and their interaction with the path being followed.

Many robot control tasks will involve planning interactions with continuous state spaces (such as navigation through 3-D space). A difficulty in the use of existing standard PDDL planners in this context is that the problem file must be complete and finite before planning starts, so it is impossible to construct a continuous state space in which arbitrary values can be generated during planning. Therefore, values of (finitely many) points within the continuous space that could be relevant to planning must be identified and named in the initial state. It is this that motivates our use of Probabilistic Roadmaps to generate a set of relevant waypoints prior to planning.

The final state in the integration of a planner is the

interpretation of plans. Plans produced by a planner are symbolic structures constructed from the actions in the domain model and instantiation with (symbolic) values from the problem instance. The symbolic constants can be interpreted by looking up values associated with them during the problem construction, as noted above, and the actions can be executed by invoking appropriate control systems to achieve their effects. There are several challenges in this process. Actions used by the planner might correspond to context-dependent behaviour in execution, so the translation of a plan into action might require monitoring of the state in order to determine the appropriate behaviour as an action is dispatched. Plans might have to be modified to include actions that were not exposed to the planner (as noted above), such as a change in pitch or yaw before moving a robot.

B. Action Dispatch

Action dispatch is an important part of the integration between planning and control levels. In our earlier work in this area [2], we used a non-temporal model and dispatched actions using a common strategy: dispatch on completion. This strategy treats a plan as a list of actions and starts each action on completion of the preceding action. In the current work we move to a temporal domain model and temporal planner. This offers an opportunity for a different dispatch strategy: dispatch on time. Using this strategy, each action is dispatched at the time assigned to it in the plan. There are interesting questions about robust dispatching of actions [14]. In particular, as actions translate into physical control, the uncertainty in the environment will usually impact on the actual duration of the realisation of each action. Actions might finish earlier or later than the model predicts and this can impact on the dispatching of the plan.

A strict interpretation of the dispatch on time strategy will cause plan failure if an action overruns. Furthermore, if an action finishes earlier than predicted, the plan will fail if the vehicle cannot pause while remaining in the same state. In contrast, the dispatch on completion strategy cannot be applied if the planner requires coordinated and synchronised actions to achieve the goal, or if there are actions that must be performed at specific times.

In our setting, pause-in-state is possible, but can be unnecessarily costly in energy terms (maintaining position can require use of thrusters to hold the vehicle against disturbances). We currently use a dispatch on completion strategy to avoid this cost, but the introduction of external events into our problem will require us to adapt this to a hybrid strategy, in which a mixed used of dispatch on completion and dispatch on time can be exploited to ensure a plan is correctly synchronised, while remaining as efficient as possible.

III. Planning AUV Inspection Tasks

We consider the situation in which an AUV has to inspect a collection of structures on the oilfield, beginning with a coarse map. Inspection points are constructed for these structures, either by recognising their forms and selecting pre-determined inspection points for them or from an ontological model available within the PANDORA architecture. The AUV is required to navigate through the environment, which is possibly different from the coarse map, and to inspect each inspection point – possibly from a number of different angles if the inspection point is only partially visible from one.

A. Method Overview

In the initial map, waypoints are placed and connected using a *Probabilistic Roadmap* [10], [11] (PRM), which is then extended with a set of *strategic* waypoints. The extended roadmap is cast as a *temporal planning problem*, and a temporal planner is used to construct a plan that will allow the AUV to move between these waypoints, visiting all the relevant structures and performing observations. Figure 2(a) shows a 2-D example of a coarse map and a PRM. As the AUV explores the environment, sonar data will reveal more detail about the precise shape and relative position of the structures. As more detail is revealed, structures in the map can be replaced with more accurate representations, the planning model can be updated and the planner can be used again to replan and find better plans. Figure 2(b) shows an example where the shape of the object has been updated based on new sonar data and a new plan has been found.

In the following we describe the overall approach in detail.

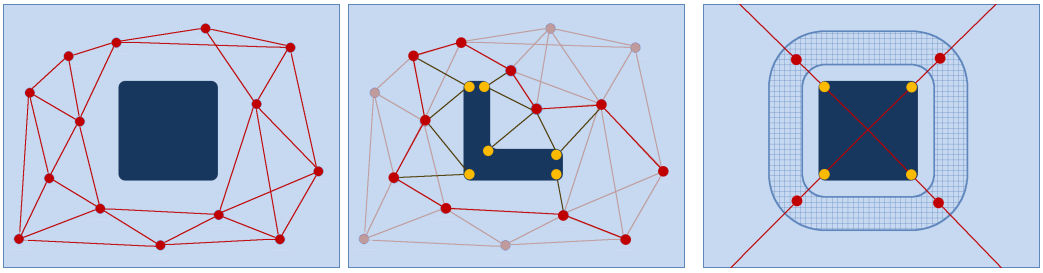
B. Roadmap Generation

The first step for planning an inspection task is to define an abstraction of the environment. To this end, a PRM is created. The planner will then use edges from the PRM in place of actual motion to approximate real distances between different locations. It is important that every edge in the PRM is representative of a *collision-free* motion for the AUV. For this reason the PRM is generated using an OMPL[1] motion planner, which takes into account the vehicle dynamics and checks for collision-free trajectories. In particular, the motion planner generates the PRM in a model of the environment provided by the PANDORA architecture and subsequently updated by sensor data.

In the current architecture the orientation of the AUV is not taken into account during planning. Indeed, explicitly modelling the orientation would add real valued variables to the atomic waypoint positions, resulting in a tricky model that would be expensive to plan with. Furthermore, the actual orientation of the AUV is variable in practice due to issues in control and that would make the plan not robust in execution. For this reason, in order to ensure edges remain collision-free when the AUV re-orients itself during the execution of a plan, we use a conservative approach, and a bounding sphere is used to approximate the shape of the AUV during collision detection.

Inspection points are only visible from certain locations. In order to ensure that the PRM contains sufficient viewpoints for each inspection point additional waypoints are added to the PRM. We call these waypoints *strategic points*. It is possible to continually make the PRM more dense, until there is sufficient coverage of each inspection point.

[1]The Open Motion Planning Library [15].



(a) Roadmap generation (b) Plan generation (c) Strategic points

Fig. 2. The coarse map for inspecting a submerged structure (dark blue) and a preliminary roadmap (red). The map of the submerged structure is updated as sonar data becomes available (b). Following the generated plan, the AUV moves through the highlighted (red) edges, and makes observations (dark lines) at some waypoints. Many inspection points require inspection from multiple locations. Inspection points (yellow) are used to generate strategic points (red) between the safest approach distance from the object and the maximum viewing distance (c).

```
(:durative-action do_hover
 :parameters (?v - vehicle ?from ?to - waypoint)
 :duration ( = ?duration (* (distance ?from ?to)
                            (invtime ?v)))
 :condition (and (at start (at ?v ?from))
                 (at start (connected ?from ?to)))
 :effect (and (at start (not (at ?v ?from)))
              (at end (at ?v ?to))))

(:durative-action observe
 :parameters (?v - vehicle ?wp - waypoint
              ?ip - inspectionpoint)
 :duration ( = ?duration (obstime))
 :condition (and (at start (at ?v ?wp))
                 (at start (cansee ?v ?ip ?wp)))
 :effect (and (at start (not (cansee ?v ?ip ?wp)))
              (at end (increase (observed ?ip)
                       (obs ?ip ?wp)))))
```

Fig. 3. A fragment of the PDDL inspection-task domain showing two durative actions, *hover* and *observe*.

```
(define (problem inspection-task-p1)
  (:objects auv - vehicle
            wp1 wp2 wp3 ... - waypoint
            ip1 ip2 ip3 ... - inspectionpoint)
  (:init
    (at auv wp1)
    (= (mission-time) 0)
    (= (observed ip1) 0)
    (connected wp1 wp2)  (connected wp2 wp1)
    (= (distance wp1 wp2) 7.16958)
    (= (distance wp2 wp1) 7.16958)
    (connected wp1 wp9)  (connected wp9 wp1)
    (= (distance wp1 wp9) 3.21484)
    (= (distance wp9 wp1) 3.21484)
    ...
    (cansee auv ip4 wp12)
    (= (obs ip4 wp12) 0.445331)
    ...
  )
  (:goal (and (>= (observed ip1) 1)
          ...
  ))
  (:metric minimize (total-time)))
```

Fig. 4. A fragment of the PDDL inspection-task problem

However, this comes at the cost of increasing the size and difficulty of the resulting planning problem. As a result, strategic points are an equally robust yet realistic alternative. Note that the PRM plays a key role as it ensures a connecting structure that links the strategic waypoints. Furthermore, the generation of the PRM is very fast, and this allows building new PRMs to take into account new information about the environment, so gaining flexibility to respond to new situations.

Strategic points are obtained by projecting rays from the detected structure and sampling along the ray. New waypoints are sampled between safest approach distance and maximum effective viewing distance away from the observed structure. This process is shown in Figure 2(c). These waypoints are connected to the Roadmap using the motion planner and can then be used when planning an inspection path.

C. A Temporal Planning Domain for the Inspection Task

In this section we briefly describe how the inspection task can be modelled as a temporal planning problem in PDDL [13]. In order to define a domain for the inspection task scenario, we consider the waypoints of the PRM generated in the previous phase. The state of the AUV is described through its position, given by a waypoint. Then, the AUV can perform two actions, namely *hover* and *observe*, as shown in Figure 3.

In PDDL, actions are described through the following components: a list of *parameters*, listed with a question mark denoting the variable tokens and the corresponding types; the *precondition* that defines the conditions that must be satisfied in the current state in order for the action to be performed; and the *effect* that defines the change in the state after the action has been performed.

The hover action moves the AUV between two connected waypoints (which, by construction, are the end-nodes of a collision-free edge). An action can be applied only if its preconditions are satisfied, therefore in this example the AUV can move from waypoint `?from` to waypoint `?to` only if it is currently at waypoint `?from`, and the two waypoints are connected. As the effect of the action, the AUV leaves the waypoint `?from` when the action starts and reach the waypoint `?to` at the end of the action. The duration of the action depends on the distance between the two waypoints. In order to take into account the time required for the AUV to turn and be in the correct orientation, we use a conservative model in which the duration of navigation actions is determined by a bound on the expected time, including corrections and reorientations. This means that the plan is robust to concurrent activity or deadlines if the vehicle could hover at a point on arrival to await the expected arrival time, assuming it arrives earlier than the conservative

bound suggests.

The observe action allows the AUV to observe an inspection point. The precondition requires the AUV to be at a waypoint from which the target inspection point is (partially) visible. Note that the effect `(not (cansee ?v ?ip ?wp))` prevents the AUV from observing the same portion of the structure more than once.

The structure to be inspected is described as a PDDL problem by encoding the generated PRM as the list of all pairs of connected waypoints together with the distance between each pair; the list of inspection points that can be observed from each waypoint (if any) together with a value indicating what percentage of each area can be observed; the initial position of the AUV; the goal state and the metric function. A fragment of a sample problem is shown in Figure 4.

D. Solving the Planning Problem

To solve the problem, we use the forward search temporal planner POPF [16]. As described earlier, the planner deals with coarse-grained events: in this case movement between waypoints and observation of inspection points. The plan generated by POPF does not specify how this movement is to take place, or what the orientation of the AUV must be in order to successfully carry out the inspection tasks, as these are the tasks for the controller at a lower level, as discussed in Section I. Figure 2(b) shows a 2-D example of a plan, while a fragment of its PDDL representation is shown in Figure 5 (left).

The plan may contain waypoints where multiple observe actions take place. In order for the plan to be valid, new motions must be introduced for re-orienting the vehicle between adjacent inspection actions. Additionally, the plan may require the AUV to revisit a waypoint, but the orientation might not sensibly be the same; for example, the AUV might be travelling in the opposite direction. For this reason a single waypoint in the task planning problem might correspond to multiple configurations of the AUV. Both of these issues are accounted for in a post processing step, that creates a file mapping the symbolic representation of the waypoints used by the planner to their real coordinates (including orientation).

IV. Plan Execution and Replanning

The planning approach described in the previous section has been implemented in a system that integrates planning and execution in the ROS framework. In ROS the computation is performed by nodes, therefore the planner is set up in a node that provides an RPC service. This service can be requested by another part of the system at any time with an environment file and a list of inspection points, as described previously.

Once the planner has found a plan and the waypoints file has been generated, they need to be converted into ROS messages to be sent to the AUV. To this aim, actions in the plan (Figure 5 - center) are tokenized and each action is translated into a ROS actionlib goal invocation. These goal invocations are passed sequentially to the controller by an executor. The controller is responsible for achieving the goal and providing feedback. The feedback can be either *success*, in which case the executor passes the next goal invocation to the controller, or *failure*, with a request for replanning. In the following we describe each step of the integration and the mechanisms behind replanning in more detail.

A. Integration and Execution

The executor reads the list of actions in the plan into a queue for sequential execution. In order for the AUV to effect an action, the symbolic descriptions of actions must be converted into a numerical form for initialization of an actionlib goal. The first token of a plan element is the action name (in this case `hover` or `observe`). Each action name is associated with a ROS actionlib goal invocation. As an example, Table 6 shows the format of the ROS actionlib goal invocation corresponding to the `observe` action. The values of each goal field are read from the waypoints file (as in Figure 5 (right)).

ROS action	goal field		value field
ObserveGoal	Pose waypoint		
		NED position	`float64` north
			`float64` east
			`float64` depth
		RPY orientation	`float64` roll
			`float64` pitch
			`float64` yaw
	NED inspectionpoint		`float64` north
			`float64` east
			`float64` depth

Fig. 6. ROS actionlib goal invocation for the `observe` action.

B. Feedback and Replanning

Fig. 7. Simulation scenario for the inspection task

If a ROS goal is achieved successfully, the executor takes into consideration the next goal invocation, corresponding to the next action in the plan. It can happen, however, that the action execution fails. Actions failures are mainly due to wrong assumptions about the environment and the structures to inspect. Indeed, as we said, the inspection task mission starts with an assumption about the map of the world. The inspection points are defined according to this assumption. However, during the mission new sonar data becomes available (as a result of the `observe` action) that can reveal that the real environment does not match the expected one. In such a case, the world map is updated and the `observe` action declares the execution to have failed. The fail signal causes the executor to request a replan on the updated world map.

Note that, in our framework, *replanning* is based on *reformulating* the inspection task as a new planning problem.

Original Plan	Post-processed Plan	Waypoint coordinates (x,y,z,roll,pitch,yaw)
0.00: (hover auv wp1 wp14) [40.38] 40.38: (hover auv wp14 wp12) [36.97] 77.35: (observe auv wp12 ip4) [10.00] 87.35: (observe auv wp12 ip3) [10.00] 97.35: (observe auv wp12 ip1) [10.00] 107.35:(observe auv wp12 ip2) [10.00] 117.35:(observe auv wp12 ip5) [10.00] 127.35:(hover auv wp12 wp10) [17.48] 144.84:(observe auv wp10 ip2) [10.00] 154.84:(observe auv wp10 ip4) [10.00]	0.00: (hover auv wp1 wp14) [40.38] 40.38: (hover auv wp14 wp12) [36.97] 77.35: (observe auv wp12 ip4) [10.00] 87.35: (hover auv wp12 wp12a) [1.00] 88.35: (observe auv wp12a ip3) [10.00] 98.35: (hover auv wp12a wp12b) [1.00] 99.35: (observe auv wp12b ip1) [10.00] 109.35: (hover auv wp12b wp12c) [1.00] 110.35: (observe auv wp12c ip2) [10.0] 120.35: (hover auv wp12c wp12d) [1.00] 121.35: (observe auv wp12d ip5) [10.00] 131.35: (hover auv wp12d wp10) [17.48] 149.84: (observe auv wp10 ip2) [10.00] 159.84: (hover auv wp10 wp10a) [1.00] 160.84: (observe auv wp10a ip4) [10.00]	wp1 8 0 0 0 0 0 wp14 4.35 -1.74 0 0 0 -2.35 wp12 1.74 -4.35 0 0 0 1.95 wp12a 1.74 -4.35 0 0 0 1.34 wp12b 1.74 -4.35 0 0 0 1.18 wp12c 1.74 -4.35 0 0 0 2.54 wp12d 1.74 -4.35 0 0 0 2.32 wp10 0 -4.5 0 0 0 2.46 wp10a 0 -4.5 0 0 0 1.10

Fig. 5. Plan generation for an inspection task. An initial plan is found using POPF (left). The plan is then post-processed (center). Each waypoint is associated with its coordinates (right).

The re-modelling is performed dynamically, as new information becomes available, the current PRM is then updated according to the new information about the environment and the structures to inspect, and then a new plan is generated.

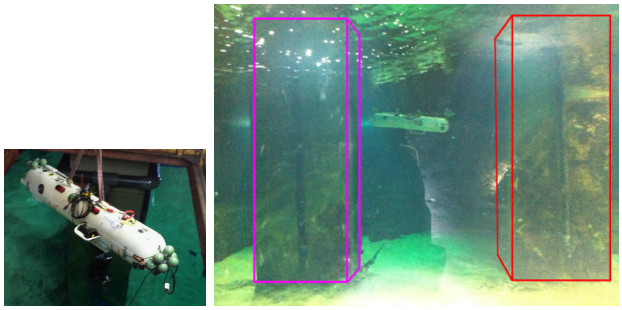
Time (s)	
PRM Construction	Planning
0.67	0.71
0.94	0.56
0.67	1.04
0.67	0.85

Fig. 8. Computation time for generating the roadmap and planning.

The processes of generating the roadmap and finding a plan are both very fast, as shown in Table 8, where we report the computation time required for constructing the roadmap and the plan in different tests. The overall approach was tested in simulation. A screenshot of the simulation is shown in Figure 7 and more details can be found in [2].

V. Physical Trials

The planning loop was evaluated on NESSIE VII at the Fort William Diver and ROV driver Training Center, Scotland. The purpose of the tests was to find logical flaws in the system that may not have been apparent in the simulated environment and to demonstrate that the temporal domain produced better plans than a non temporal formulation. NESSIE VII AUV has six thrusters, a Doppler Velocity Log and a fibre optic gyro which forms a closed loop positioning control system (see Figure 9(a)) . The AUV also is fitted with a forward sonar, and a micro bathymetry sonar.



(a) Nessie VII AUV (b) Real scenario for the inspection task

Fig. 9. Setup for the physical trials

A. Map Construction

The underwater tank at Fort William contains two pillars. The pillars are roughly one metre wide, two metres thick, and just over three metres apart (see Figure 9(b)). Both pillars have a pipe fixed vertically to all four sides. The physical test mission was to inspect each of these pipes. We modelled the environment as a COLLADA file, and defined a set of inspection points. The AUV was placed in a prescribed starting location to the North-East of the pillars.

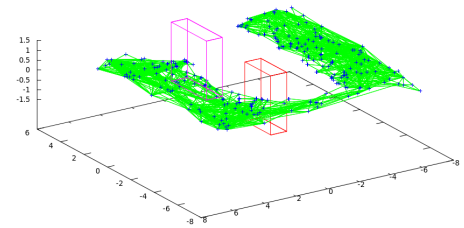
B. Mission Planning

Once the map of the inspection task scenario has been made, we then moved to apply the process described in Section III. Namely, we provided the motion planner with the COLLADA map file and the set of inspection points. Then, we used the motion planner to generate the probabilistic roadmap shown in Figure 10(a), which includes the strategic points. Finally, we used the temporal planner POPF to find the optimised plan. The path of the produced plan is shown in Figure 10(b). This step was run several times, with the temporal domain and also with a non temporal domain in which time and concurrency are not considered. The times to complete these plans were estimated using the distance of the path and the preset time for an observation action.

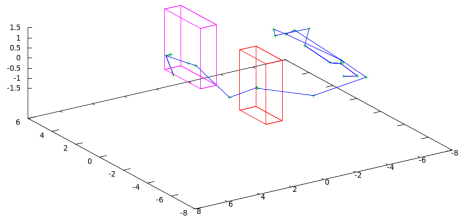
C. Plan Execution and Mission Outcome

The execution of a hover action results in a new desired position being sent to the closed loop position control system. This loop decides how much thrust to apply at each thruster location in order to move the robot towards the desired position. Like many PID control systems, the control system tends to reduce thruster power as the AUV is near to destination location, and subsequently may never reach the desired position exactly. We configured the system to decide the hover action to be completed when the robot was within 0.3m of the desired location.

The AUV successfully completed the plan as it was described by POPF, finishing by returning to the initial position. Logical flaws in the system, not apparent in simulation, were discovered during these tests. Some physical capabilities of the AUV were not modelled precisely in the planning domain. Specifically, the AUV was not able to move close to the surface, as the resulting bubbles interfered



(a) The PRM generated for the environment shown in figure 9(b).



(b) The path of the plan generated using the PRM (a).

Fig. 10. Planning inspection task

with the DVL. This was simple to fix by reducing the height of the bounding box used during PRM generation, thus keeping the AUV 0.3m under the surface at all times.

The estimated times for the plans generated in the mission planning phase are shown in Table 11. The results make clear the befenits of using a temporal model, as it provides a significant reduction of the time needed to complete the mission. In particular, the use of temporal planning in this domain allows a significant optimisation, as the planner chooses not only the shorter paths between inspection targets but also a reduced number of observations by choosing waypoints with greater visibility of the inspection point. In fact, Table 11 also shows that using the temporal domain can significantly reduce the number of observations, often in this scenario by the number of inspection points. These estimations are representative of the real time to complete the mission. In future work we will return to this mission, using a map dynamically constructed from the sensor data of the AUV.

Time (s)		Additional
Temporal	Non Temporal	Observes
276.393	1026.565	0
345.609	480.96	8
228.304	401.092	9
286.496	500.478	9

Fig. 11. Estimated time and additional observe actions for missions with the temporal and non-temporal models. The third column states the number of extra observe actions taken by the plan in the non-temporal domain.

VI. Conclusions

The process we have decribed here combines task planning in a temporal domain with controllers in a feedback loop. Path planning is used to generate a set of waypoints and accessibility network as a basis for task planning, which then constructs an inspection trajectory by selecting the subset of waypoints to visit in order to gain sufficient coverage of the inspection points, while navigating between them using the paths identified by the path planner. Although others have also considered the problem of planning inspections using AUVs and also integration of path and task planning we also frame the problem as a temporal planning problem. The use of a temporal domain ensures that the resulting plan is optimised with respect to duration, and is capable of more sophisticated behaviours in more complex missions. We also report on physical trials that took place in a large underwater tank at Fort William, Scotland, succesfully demonstrating the effectiveness of the framework described in this paper, and the benefints of using a temporal model. This represents a significant new development in the exploitation of task planning for controlled deployment of AUVs.

Acknowledgments

This work was partially funded by the EU FP7 Project 288273 PANDORA.

References

[1] "Robot Operating System (ROS)," www.qbflib.org, last accessed Jul. 2013.

[2] M. Cashmore, M. Fox, T. Larkworthy, D. Long, and D. Magazzeni, "Planning inspection tasks for auvs," in *Proceedings of OCEANS'13 MTS/IEEE*, 2013.

[3] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. S. McEwen, "A deliberative architecture for auv control," in *ICRA*, 2008, pp. 1049–1054.

[4] F. Py, K. Rajan, and C. McGann, "A systematic agent framework for situated autonomous systems," in *AAMAS*, 2010, pp. 583–590.

[5] E. Hernández, M. Carreras, and P. Ridao, "A path planning algorithm for an auv guided with homotopy classes," in *ICAPS*, 2011.

[6] E. Hernández, M. Carreras, J. Antich, P. Ridao, and A. Ortiz, "A topologically guided path planner for an auv using homotopy classes," in *ICRA*, 2011, pp. 2337–2343.

[7] B. Englot and F. Hover, "Inspection planning for sensor coverage of 3D marine structures," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2010.

[8] E. Plaku and J. McMahon, "Combined mission and motion planning to enhance autonomy of underwater vehicles operating in the littoral zone," in *Workshop on Combining Task and Motion Planning at IEEE International Conference on Robotics and Automation (ICRA'13)*, 2013.

[9] M. Fox, D. Long, and D. Magazzeni, "Plan-based policy-learning for autonomous feature tracking," in *ICAPS*, 2012.

[10] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[11] S. M. Lavalle, *Planning Algorithms*. Cambridge, 2006.

[12] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[13] M. Fox and D. Long, "PDDL2.1: An extension to pddl for expressing temporal planning domains," *J. Artif. Intell. Res. (JAIR)*, vol. 20, pp. 61–124, 2003.

[14] M. Fox, R. Howey, and D. Long, "Exploration of the robustness of plans," in *AAAI*, 2006, pp. 834–839.

[15] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.

[16] A. J. Coles, A. I. Coles, M. Fox, and D. Long, "Forward-Chaining Partial-Order Planning," in *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, 2010.