Planning Inspection Tasks for AUVs

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. In the EU FP7 project 288273, PANDORA, we seek to perform these tasks using autonomous underwater vehicles, 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. Although we assume that some knowledge of the environment is given (a blueprint of the expected structures), we consider the problems of adapting to discoveries of modified structures (which could be a consequence of unrecorded engineering work or of shifting environmental conditions) and both planning and replanning suitable inspection missions. We demonstrate that these missions can be performed in a simulation and that performance is suitable for deployment in the proposed setting.**

I. Introduction

We are exploring the problem of autonomous inspection and maintenance of a seabed facility, in the context of the EU FP7 project 288273 PANDORA. The autonomous inspection and maintanence is to be carried out using AUVs over extended horizons. To achieve this requires extensive operation without human intervention, so it demands not only sophisticated controllers to manage the necessary navigation and interaction tasks, but also planning, to assemble the lower level tasks into a coherent mission plan over a rolling horizon.

The project seeks to examine a range of issues in the integration of planning and execution, including different aspects of the problems that arise because of the uncertainty in the execution environment. However, we have, at this stage, focussed on the problem of 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.

We have implemented a system that integrates planning and execution in the Robot Operating System (ROS) framework [1]. The ROS framework allows us to run our system in simulation with the underlying vehicle dynamics, and provides functionality for integrating the various levels of our architecture.

The use of AI Planning in AUV mission control is not new. Recent works include [2] (where homotopy classes are used to guide the path planning), [3] (where a coarse plan is generated to initialise the inspection of an unknown hull), [4] (where sampling-based motion planning is used to solve missions modelled in LTL), or [5] (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 way of modelling the AUV behaviour in PDDL [6], which is based on an abstraction of angles and orientations which are represented symbolically;
- we show how a probabilistic roadmap can be used to generate a finite map as an input to a standard PDDL planner;
- we present a new approach for modelling the environment for the inspection task, based on extending the standard probabilistic roadmap with a set of strategic points;
- we present a new replanning strategy which is based on a dynamic re-modelling of the environment.

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, as we show in the following sections.

Our architecture is similar to that described by Plaku and McMahon [4] in that the high-level reasoning is carried out on an abstraction of the problem, provided by a roadmap. In contrast to that approach, we use a planner at this level to reason about the order of inspection tasks and high-level motion. The output of the planner is carried out at lower levels.

The paper is structured as follows. After a brief discussion of planning and control integration in Section II we provide an overview of the problem and our method in Section III. Section IV explains our method in more detail, focussing on the role that planning has in the overall architecture. Section V explains how the planning problem integrates with the lower-level aspects of the system, how plans are carried out at the control level, and describes the role of feedback between these levels. Section VI provides the results of our system running in simulation using ROS. Finally, we conclude in Section VII.

II. Integrating Task Planning and Control

Planning is a technology explored in Artificial Intelligence that seeks to organise activities in order to achieve specific goals [7]. It begins with a domain model describing the actions available to the planner, in terms of their pre- and post-conditions, and a description of the current state (of both the AUV and its environment). Faced with goals to achieve it constructs a plan by organising instances of the actions into a structure that is causally valid and is predicted to achieve

978-0-933957-40-4

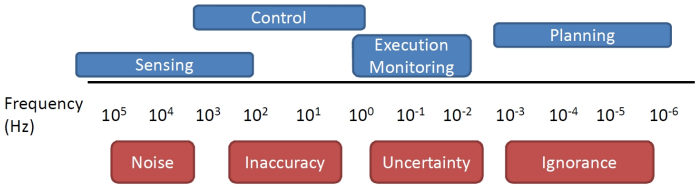

Fig. 1. Planning and control integration

the goals, while optimising a cost function. It is important to recognise that planning is completed before execution: it is not a substitute for control, providing reactive decision-making in a tight loop with sensing, but is concerned with making choices about the assembly of actions, forecasting their interaction with current and future constraints to avoid locking the executive into inescapable problems, while achieving goals efficiently.

Planning is a high level reasoning activity that sits above several more fundamental components such as sensing and control. 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 much more coarse-grained activity than the control systems that it deploys. Control systems are deployed to help to manage disturbances in behaviours. 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. The planner adapts to unexpected features by revising its abstract, deterministic model of the world and replanning to take account of the new features.

The challenges in connecting task planning and control have been considered in much prior work.

One of the design principles in 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) 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, the planner would

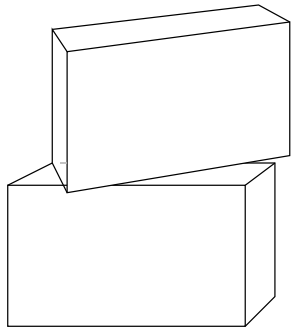

Fig. 2. Crates satisfying $on(c_1, c_2)$.

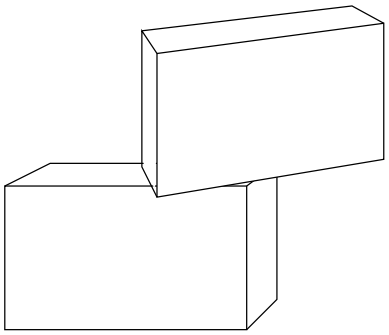

Fig. 3. Crates satisfying $on(c_1, c_2)$ in an alternative configuration.

have to add them to the plan, but might embark on unnecessary search in a futile attempt to avoid using them. 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 will generally involve some kind of abstraction process, translating sensor data into symbolic representations. In addition, because the planner will manipulate symbolic representations of objects, it is often useful to store data associated with the interpretation of those objects 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). To see the significance of this, consider a simple symbolic predicate, *on(A,B)*, intended to capture that object A is on top of object B. Suppose objects c_1 and c_2 are identically shaped cuboid crates; we would probably expect a situation in which c_2 is on a flat, horizontal surface and c_1 is resting entirely on c_2, although not aligned with it (Figure 2) to satisfy $on(c_1, c_2)$. However, if we start with the crates on the ground and ask for a plan to satisfy $on(c_1, c_2)$, we would probably not be entirely satisfied with a

plan that led to a state such as that shown in Figure 3. This is a difficult problem to overcome and solutions currently depend on the way in which actions are executed, which we discuss shortly.

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. Plans are produced as sequences of actions (in classical planning) or a time-stamped actions with duration (in temporal planning): dispatching either of these forms of plans requires that decisions be made about the reaction to actions completing later or earlier than expected. In particular, whether or not to simply continue by dispatching the next action in the plan, or to wait for the expected dispatch time, or to consider the plan failed. In our context, we receive plans as sequences of actions and dispatch them in sequence, starting the next when its predecessor completes, regardless of timing.

III. AUVs Inspection Tasks

A. Problem

We consider the situation in which an AUV has to inspect a collection of structures on the oilfield, beginning with a coarse map, a 2-D slice of which is shown by the dark blue shape in Figure 4. Here, the precise shapes and conditions of the structures are not known, so the structures are represented as simple shapes surrounded by free space.

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. Inspection points are represented by yellow markers in both Figure 6 and Figure 5.

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.

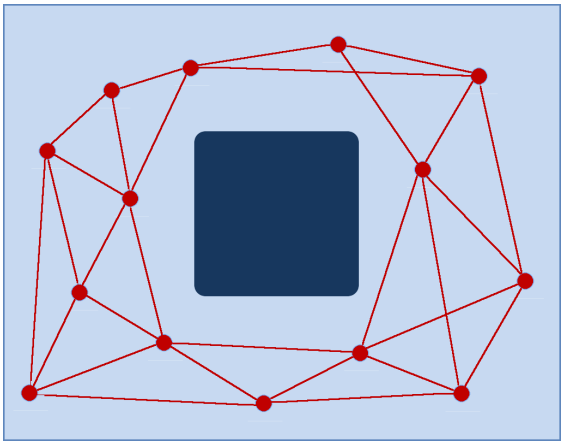

Fig. 4. The coarse map for inspecting a submerged structure (dark blue) and a preliminary roadmap (red).

B. Method Overview

In the initial map, waypoints are placed and connected using a *Probabilistic Roadmap* [8], [9] (PRM), which is then extended with a set of *strategic* waypoints. The extended roadmap is used to generate the input for the planner to construct a plan that will allow the AUV to move between these waypoints, visiting all the relevant structures and performing observations. Figure 4 shows a 2-D example of a coarse map and a PRM. We call this the fly-past plan and it plays a similar role to the initial survey used in [3] to initialise a model for hull inspection of an unknown hull. As the AUV executes its fly-past plan, 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, and the planner can be used again to replan and find better plans. Figure 5 shows an example where the shape of the object has been updated based on new sonar data and a new plan has been found.

The inspection plans are built using the planner POPF [10], and the actions in the plan are sent in sequence to a controller using the ROS framework. The planning phase is described in detail in Section IV, while the integration within ROS and replanning are described in Section V.

IV. Planning Inspection Tasks

A. 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.

[1]The Open Motion Planning Library [11].

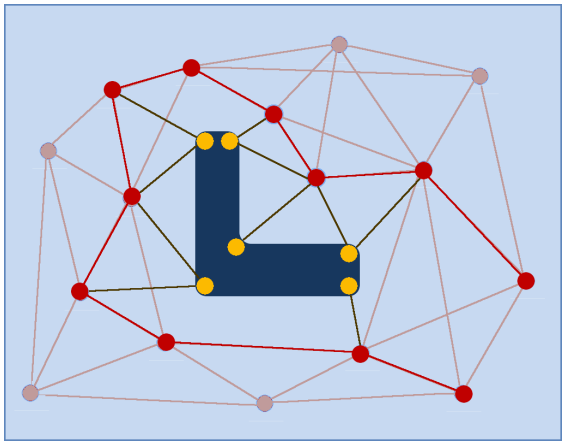

Fig. 5. The map of the submerged structure is updated as sonar data becomes available. Following the generated plan, the AUV moves through the highlighted (red) edges, and makes observations (dark lines) at some waypoints. Many inspection points require inspection from multiple locations.

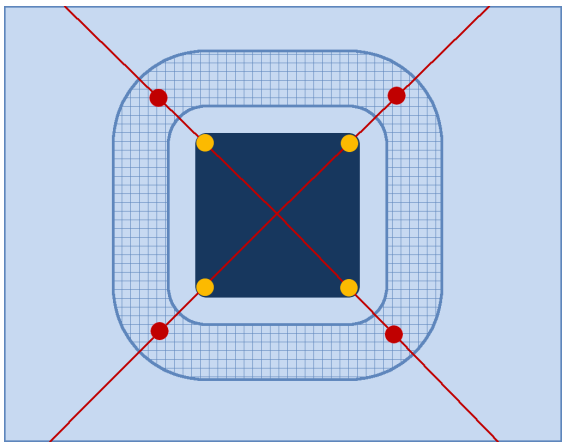

Fig. 6. Inspection points (yellow) are used to generate strategic points (red) between the safest approach distance from the object and the maximum viewing distance.

In the current architecture the orientation of the AUV is not taken into account during planning. 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. 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.

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 6. These waypoints are connected to the Roadmap using the motion planner and can then be used when planning an inspection path.

B. Inspection Task Planning Model

In this Section we describe how the inspection task can be modelled as a planning problem in PDDL [6], which is the standard language for describing planning problems. In Section IV-C we show how a planner can be used to solve it. In PDDL, a planning model consists of two components:

- a planning *domain*, containing the description of the actions the planner can choose; and
- a planning *problem*, describing the initial state and the goal to be achieved.

Clearly, given a particular domain, a planner can be used to solve many problems on the same domain.

1) Planning Domain: 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 7.

```
(:action hover
 :parameters (?v - vehicle
              ?from ?to - waypoint)
 :precondition (and (at ?v ?from)
                    (connected ?from ?to))
 :effect (and  (not (at ?v ?from))
               (at ?v ?to)
               (increase (mission-time)
               (* (distance ?from ?to) 10)))
)

(:action observe
 :parameters (?v - vehicle ?wp - waypoint
              ?ip - inspectionpoint)
 :precondition (and (at ?v ?wp)
                    (cansee ?v ?ip ?wp))
 :effect (and (increase (observed ?ip)
                        (obs ?ip ?wp))
              (not (cansee ?v ?ip ?wp))
              (increase (mission-time) 2))
  )
)
```

Fig. 7. A fragment of the PDDL inspection-task domain.

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). In particular, the AUV can move from waypoint `?from` to waypoint `?to` only if it is currently at waypoint `?from`, and the two waypoints are connected (i.e.,

the predicate (connected ?from ?to) is true). As the effect of the action, the AUV will be at waypoint ?to and no longer at waypoint ?from. Furthermore, the duration of the mission is incremented by the duration of the action (which depends on the distance between the two waypoints).

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 (predicate cansee). The action effect increases the duration of the mission and sets the portion of the inspection point that has been inspected. Note that the effect (not (cansee ?v ?ip ?wp)) prevents the AUV from observing the same portion of the structure more than once.

2) Planning Problem: each inspection-task mission can be described as a planning problem for the inspection-task domain. Figure 8 shows a fragment of a sample problem where all the relevant components can be observed. In particular, the problem contains the following elements: the set of objects describing the initial state (i.e., the waypoints wp_i and the inspection points ip_j); the generated PRM, described 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; and the initial position of the auv (i.e., (at auv wp1)).

Finally, the problem contains the description of the goal state, that, in the example of Figure 8, consists of completely observing inspection points $\text{ip}_1 \ldots \text{ip}_5$.

C. Solving the Planning Problem

To solve the problem, we use the forward search planner POPF [10]. 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 II. Figure 5 shows a 2-D example of a plan, while a fragment of its PDDL representation is shown in Figure 9 (left).

The plan may contain waypoints where multiple observe actions take place. As mentioned, the planner does not take the orientation of the AUV into account. 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.

This step produces a *mapping* file containing coordinate data for use by the controller. Each waypoint in the post-processed plan corresponds to a coordinate, including orientation, in the file.

The post-processing is performed as follows:

1) New movement actions for re-orientation are inserted between adjacent inspection actions.

```
(define (problem inspection-task-p1)
  (:domain inspection-task)
  (:objects
    auv - vehicle
    wp1 wp2 wp3 wp4 wp5 wp6 ... - waypoint
    ip1 ip2 ip3 ip4 ip5  - inspectionpoint
  )
  (:init
    (at auv wp1)
    (= (mission-time) 0)
    (= (observed ip1) 0)
    (= (observed ip2) 0)
    (= (observed ip3) 0)
    (= (observed ip4) 0)
    (= (observed ip5) 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)
 ...

    (connected wp10 wp9)  (connected wp9 wp10)
    (= (distance wp10 wp9) 3.52376)
    (= (distance wp9 wp10) 3.52376)

    (cansee auv ip4 wp12)
    (= (obs ip4 wp12) 0.445331)
 ...

    (cansee auv ip2 wp10)
    (= (obs ip2 wp10) 0.418531)
  )
  (:goal (and
    (>= (observed ip1) 1)
    (>= (observed ip2) 1)
    (>= (observed ip3) 1)
    (>= (observed ip4) 1)
    (>= (observed ip5) 1)
  ))

  (:metric minimize (mission-time))
)
```

Fig. 8. A fragment of the PDDL inspection-task problem.

2) Waypoints are duplicated for each time they are revisted in the plan. The duplicate waypoints share the same position as the original, but will have possibly different orientation[2]. Every time the plan requires the AUV to revisit a waypoint – including during re-orientation – an unused duplicate waypoint is used instead.
3) Orientation is generated for each waypoint, pointing the AUV either in the correct direction for a subsequent inspection action, or towards the next waypoint in the path.
4) The positional and orientational information for each waypoint is printed to the mapping file in the order they are visted in the plan.

[2]In Figure 9 waypoints 12, 12a, 12b, 12c, 12d are duplicates, as well as waypoints 10 and 10a.

Original Plan	Post-processed Plan	Waypoint coordinates (x,y,z,roll,pitch,yaw)
0: (hover auv wp1 wp14) 1: (hover auv wp14 wp12) 2: (observe auv wp12 ip4) 3: (observe auv wp12 ip3) 4: (observe auv wp12 ip1) 5: (observe auv wp12 ip2) 6: (observe auv wp12 ip5) 7: (hover auv wp12 wp10) 8: (observe auv wp10 ip2) 9: (observe auv wp10 ip4)	0: (hover auv wp1 wp14) 1: (hover auv wp14 wp12) 2: (observe auv wp12 ip4) 3: (hover auv wp12 wp12a) 4: (observe auv wp12a ip3) 5: (hover auv wp12a wp12b) 6: (observe auv wp12b ip1) 7: (hover auv wp12b wp12c) 8: (observe auv wp12c ip2) 9: (hover auv wp12c wp12d) 10: (observe auv wp12d ip5) 11: (hover auv wp12d wp10) 12: (observe auv wp10 ip2) 13: (hover auv wp10 wp10a) 14: (observe auv wp10a ip4)	wp1 8 0 0 0 0 0 wp14 4.35695 -1.74278 0 0 0 -2.3562 wp12 1.74278 -4.35695 0 0 0 1.9513 wp12a 1.74278 -4.35695 0 0 0 1.34894 wp12b 1.74278 -4.35695 0 0 0 1.18361 wp12c 1.74278 -4.35695 0 0 0 2.54645 wp12d 1.74278 -4.35695 0 0 0 2.32449 wp10 0 -4.5 0 0 0 2.46685 wp10a 0 -4.5 0 0 0 1.10715

Fig. 9. 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).

Therefore, the post-processing step creates a file mapping the symbolic representation of the waypoints used by the planner to their real coordinates.

V. 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 9 - 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. The state diagram for the executor is summarised in figure 10.

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 11 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 9 (right)).

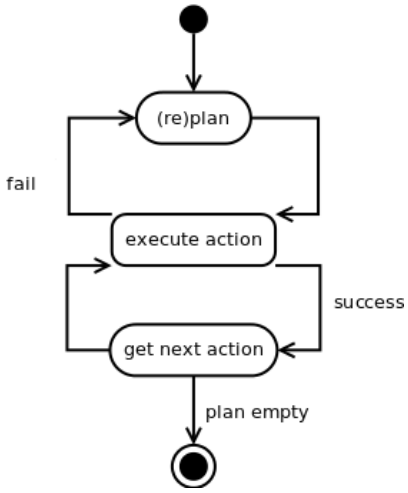

Fig. 10. State diagram of the executor

Ros actionlib 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. 11. ROS actionlib goal invocation for the `observe` action.

B. Feedback and Replanning

If a ROS goal is achieved successfully, the executor takes into consideration the next goal invocation, corresponding to

Fig. 12. Simulation scenario for the inspection task

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. 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.

VI. Simulation

A simple scenario was devised to test the replanning loop, using UWSim [12]. The AUV was told to inspect a cube shaped object, whereas in fact the cube had an unaccounted for dent on the distal side (as shown in Figure 12).

The aim of the simulation was to test the planning integration and the behaviour of the planning strategy. Therefore, at this stage we did not test the interpretation of sonar data to update the world map, instead we customised the implementation of the `observe` action so that the feedback was success if executed when the AUV was on the non-dented sides of the cube, and failure on the dented side. Furthermore, when the observe action failed on the dented side, the world map was replaced with the correct version. Testing the sonar data interpretation is out of the scope of the present paper, and will be the focus of future tests.

Figure 13 (above) shows the probabilistic roadmap generated with the motion planner. The roadmap is then given as input to the planner that finds the plan shown in Figure 13 (below).

The processes of generating the roadmap and finding a plan are both very fast, as shown in Table 15, where we report the computation time required for constructing the roadmap and the plan in different tests.

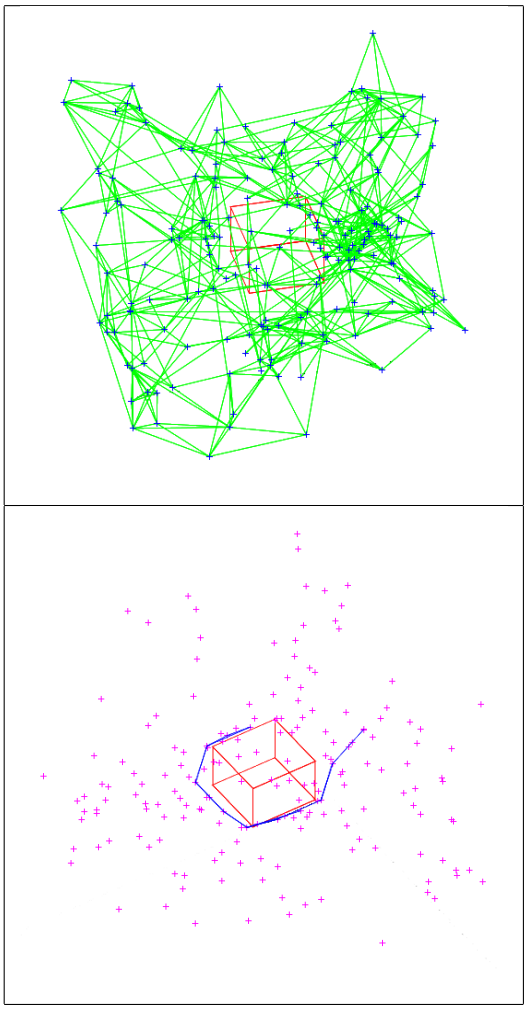

Fig. 13. Inspection task simulation: roadmap generation and planning.

Figure 14 shows the execution of a different plan in the ROS environment. The purpose of this was to test the use of strategic points. Inspection points in the structure are set in such a way that a simple trajectory over the strategic points would allow completing the inspection task.

Figure 14 (left) shows the AUV in its initial position, and its initial (incorrect) assumption that a cube is to be inspected. Around the cube are the orientated (strategic) waypoints (in green) that the AUV will use for planning hover actions between. Figure 14 (center) shows the AUV where approaching

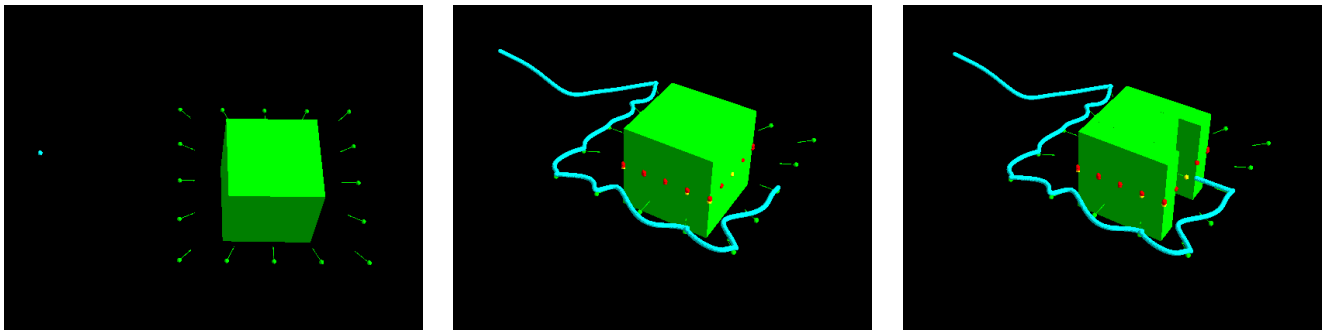

Fig. 14. Inspection task simulation.

Time (s)	
PRM Construction	Planning
0.67	0.71
0.94	0.56
0.67	1.04
0.67	0.85
0.74	0.48
0.67	0.75
0.77	0.69
0.68	0.78
0.72	0.52

Fig. 15. Computation time for generating the roadmap and planning.

the dented location. It is is attempting to look for the inspection point marked in yellow, but the sonar does not see it (red). Figure 14 (right) shows the updated map, and the corresponing updated plan, with the AUV now inspecting the dent and the remaining inspection points.

VII. Conclusion

In this paper we have concentrated on the integration of planning with path-planning, execution and control, for AUVs performing inspection tasks. The approach we have described is sufficiently general to allow us to plan a variety of other tasks in the same setting. The most important elements of our approach are the techniques for integration of standard planning systems into the architecture of an executive system, generating PDDL problem instances for domains that involve navigation in continuous space. The process we have described combines task planning with path-planning by using a sampling based path-planner 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. Ultimately, navigation is completed using the paths identified by the path planner. Furthermore, we consider the problem of structure discovery and replanning in response to a changing model of the environment. This combination represents a significant new development in the exploitation of task planning for controlled deployment of AUVs. Next steps will include the use of a temporal planning domain for modelling the inspection task and the implementation of the framework on-board the AUV for testing the approach at sea.

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

[3] 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.

[4] 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.

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

[6] 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.

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

[8] 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.

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

[10] 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.

[11] 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.

[12] M. Prats, J. Perez, J. Fernandez, and P. Sanz, "An open source tool for simulation and supervision of underwater intervention missions," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 2577–2582.