Planning Autonomous Underwater Reconnaissance Operations

Sara Bernardini
Department of Computer Science
Royal Holloway, University of London
Egham, Surrey, UK, TW20 0EX
sara.bernardini@rhul.ac.uk

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

Abstract

In this paper, we focus on the task of reconnaissance in an underwater setting. The objective is to efficiently search a large area in order to identify safe passages for ships. A path is considered safe if there are no rocky regions within it as these can conceal hazards such as mines. Our approach is to decompose the area into regions which can then be allocated for seach to individual vehicles. An initial, coarse, lawnmower survey gives some indication of the make-up of the area to be searched, and the next step is to use this survey to hypothesise about the structure of the unseen areas. It is hypothesised that any as-yet unseen areas are rocky and the task of the AUVs is to disprove this hypothesis by planning efficient paths to investigate these unseen areas. We have developed a plan-based approach that searches a large area very efficiently by reasoning about the structure of the seabed as the planned search progresses. To demonstrate the potential of our technique, we have developed a simulation based on ROS and integrated our system into the commercial "SeeByte Neptune" simulator.

1 Underwater Reconnaissance Problem

The current state of the art is that underwater operations are typically scripted or remotely operated. The successful proprietary SeeByte Neptune system (SeeByte 2013), for example, offers a way to support human planning of underwater missions at a level of abstraction, but remains demanding of human planners and offers limited opportunities for dynamic replanning on-board as missions unfold. There is a limited application of automated planning in multi-domain unmanned platforms, see for example (Cashmore et al. 2013) and (Faria et al. 2014). The first paper concerns the use of autonomous AUVs to perform inspection and maintenance tasks in underwater installations, while the second paper explores the role of planning technologies in underwater missions and considers broader-based missions with mixed platforms including UAVs and AUVs.

In this paper, we describe our approach to use generic planning technology to underpin fully autonomous reconnaissance missions in an underwater setting. In particular, we consider an underwater area A, which is initially completely unknown and where mines might be present. The seabed in this area can be: flat (F), rippled (R), mixed (M) and complex (C). We assume that the AUVs available for the reconnaissance missions are capable of detecting these features with some probability error, which is given for each feature. When a region in A is directly observed as flat is classified as F1, if a region is adjacent to an F1 region is classified as F2. The same holds for the other types of seabed. The probability of spotting a mine is different depending on the seabed quality and these probabilities are given. Flat regions offer the best visibility. Let us know call *corridor* a passage in the area A from two given points. The goal of a reconnaissance mission is to find a *safe* corridor, i.e. a passage with the lowest probability of having mines in it (see Figure 1). The ideal outcome would be to find a corridor where the seabed is flat and no mines are spotted. Mission time needs to be minimised so, although multiple observations of the same locations are possible, the information that is acquired in doing that must be balanced against the time used for those additional observations.

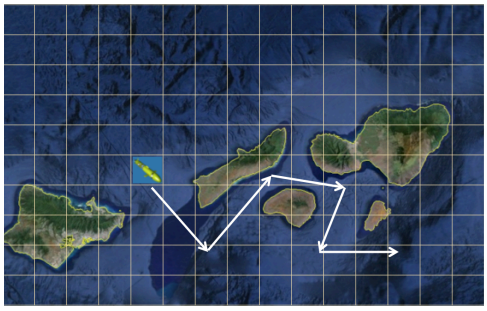

Figure 1: Underwater Reconnaissance Problem: finding a safe passage from two given locations.

Currently, in real applications (SeeByte 2013), the problem is solved as follows. First, the AUV proceeds along one diagonal d_1 in the quadrangular area A and, in doing so, classifies n quads that can be identified within A and whose diagonals overlap d_1. The classes are: F1, F2, R1, R2, M1, M2, C1 and C2. Then, the AUV does the same along one side of A and along the other diagonal d_2 (see Figure 2). This procedure is repeated until the entire area A has been classified. After classification, a corridor is identified. This method works, but is inefficient since it requires that the entire area is classified in order to find a corridor, although this is not a requirement of the reconnaissance problem itself.

We propose a new approach to underwater reconnaissance

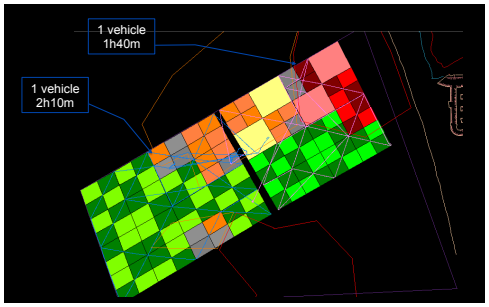

Figure 2: Classification of the underwater area to find a safe corridor. This solution is currently adopted in real-world operations, for example by SeeByte Inc.

that is based on the use of *automated planning* techniques. Our approach allows one to minimise the resources used by the AUV during the inspection tasks since it does not require a complete classification of the seabed in the area of operation in order to find a passage. At the same time, planning tools provide solutions that maximise the likelihood of discovering safe passages.

As a first step towards solving the general underwater reconnaissance problem, we have focused on one particular instance, which has been devised in consultation with our industrial collaborators at SeeByte Inc. We consider an underwater quadrangular area Q (around 2 Km x 2 Km), which is initially completely unknown. When inspected by the AUV, regions in this area can be classified as flat (F) or non-flat. We call complex (C) any region that is non-flat. We call *safe* corridor (or passage) in the area Q, a sequence of segments that connects the left side to the right side, where each segment traverses flat regions only.

1.1 Search as a Planning Problem

The approach we adopt in tackling underwater reconnaissance is based on our previous work on search-and-track (i.e. the problem of finding and following a mobile target) using an unmanned aerial vehicle (UAV) (Bernardini et al. 2013; Bernardini, Fox, and Long 2014; 2015; Bernardini et al. 2016). That work exploits generic technology: modelling tools, a temporal planner and an execution architecture, which have been redeployed in the underwater environment. There is no domain-specific behaviour in either application, but the work presented here for the reconnaissance problem changes the focus of search from a single mobile target to finding a structured static area on the seabed. This change has led to interesting further developments in our ideas, although still using the same underlying paradigm.

The paradigm we use in all of these application areas relies on a *hypothesise-and-test* approach. In search, the hard decisions have to be made when considering what to do when the target of search is *not* found: once the target is found, then behaviour changes (to a following behaviour in search-and-track missions, or to an exploitation phase when searching for safe passage for ships). This leads to the observation that one can usefully plan those hard decisions in advance, to optimise search, based on the *hypothesis* that each search attempt will fail to find the target. That is, by *anticipating* failure in each search, we can focus on the decisions about what to do next in order to continue the search. Therefore, we plan searches and hypothesise that they will fail to discover the target of search, making it useful to plan to continue to search further.

Although we use a *deterministic* planner to perform the planning, we do not ignore probability. We assume that the actions of the searching agent are predictable, but that the outcome of search is not. However, as noted above, a successful search will cause a change in behaviour, so we can plan purely on the assumption that each search will fail. In that case, we can use this anticipated failure to determine how the probability mass describing the expected outcome of searches moves around (exploiting correlations in the properties of locations). This is reflected in changing rewards for different locations and allows the planner to trade between resource efficiency and expected benefits in its direction of search.

1.2 Assumptions

In this work we make several assumptions. Some of these are simplifying assumptions but others reflect our limited knowledge in some specialised areas related to the domain. In general, these assumptions are not fundamental to the way we solve the problem, because we can extend and modify our models to reflect a refined understanding of the domain without changing the underlying technology. This is a key benefit in the use of our generic technology.

We have ignored the problem of localisation accuracy in the AUVs, which simplifies the problem of navigation when there are multiple vehicles in the area. We have ignored the issue of use of sonar in locations where there might be wildlife that prevents us from using sonar safely. We have also ignored the question of the accuracy of the identification of subsea terrain structures based on sonar data. We have also ignored the potential uncertainty in navigation caused by currents.

2 Three Phase Approach

We adopt a three phase approach to tackle the underwater reconnaissance problem. First, the vehicle performs a coarse survey of the area, which is simply a big *lawnmower*(that is, the vehicle passes backward and forward over the search area) search pattern to acquire initial, coarse-grained information about the area of operation. This pattern is not a classic lawnmower pattern (which comprises tightly adjacent paths to cover the entire region), but a simple crossing path to give an initial view of the space (see Figure 4). Then, a cycle of *planning* and *replanning* rounds generate plans for the AUV to efficiently explore the area of operation in more depth. During this cycle, the vehicle discards as many paths that contain complex surfaces as possible and accumulates information about flat areas. When it has acquired sufficient confidence that several flat regions can be linked together in a clear path, the AUV switches to the last phase, in which it goes along this hypothetical path to *confirm* that it is a safe passage. If this confirmation phase is successful, the problem is solved; if not, the planning and replanning cycle starts

again until a clear path is found. The three phase approach is illustrated in Figure 3.

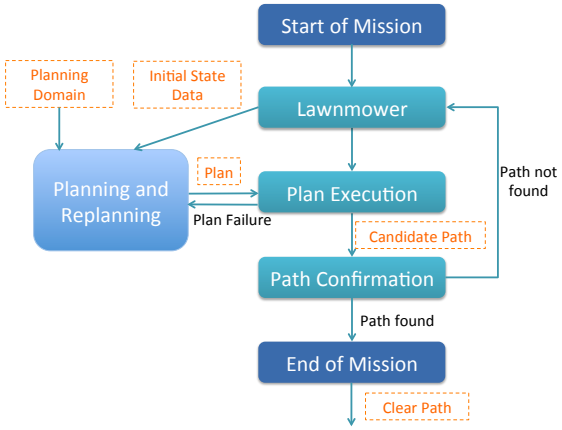

Figure 3: Three Phase Approach to the Underwater Reconnaissance Problem

3 Phase 1: Lawnmower

During the first phase, the AUV executes the initial survey pattern with three vertical legs and two horizontal legs over the area of operation Q (see Figure 4). In order to structure and store the information regarding the area of operation, we lay a grid over the area Q, where the cells have sides of 100 meters (see Figure 4). Since the AUV uses one sonar per side with a range of 50 m, we assume that the AUV is capable of observing an entire cell when it passes over it and can classify the seabed of the observed cell as flat or complex. Hence, once the lawnmower has been executed, the nature of all the cells traversed by the vehicle are known, whereas the other remain unknown (see Figure 5).

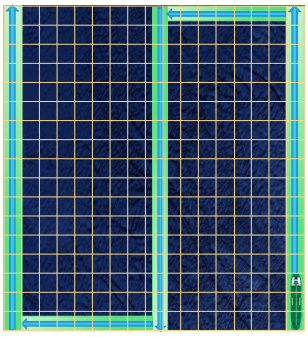

Figure 4: Grid over the area of operation and initial lawnmower (Phase 1).

4 Phase 2: Planning and Replanning Cycle

The planning and replanning cycle works in four steps:

1. Hypothesis generation;
2. Planning task formulation and planning;

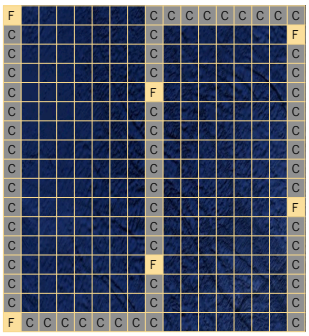

Figure 5: Initial information

3. Plan execution;
4. Replanning.

During the first step, based on the information collected during Phase 1 (see flat cells in Figure 5), we hypothesise potential clear segments in the area of operation by connecting together flat cells (see red arrows in Figure 6). These segments, appropriately combined, could form a clear passage from one side to the other side of the area of operation. Then, we ask the planner to check these hypotheses and discard the segments that contain complex surfaces since they are not suitable components of clear paths. In order to do that, we formulate the problem of disproving hypotheses as a planning task and we feed this task into an high-performing planner, called POPF-TIF (Piacentini et al. 2015). We then execute the plan provided by the planner until completion or until a plan failure occurs. In this context, a plan failure happens when the planner cannot discard a segment since, when checking it, it has not found complex cells in it. At this point, we abandon the current plan and generate a new planning problem based on the new information that we have acquired during the previous plan execution step. This replanning step provides a new plan that is in turn executed. This cycle of planning and replanning continues until we conclude that no passage exits or until we have gained sufficient information about a potential clear path, which is then validated in Phase 3. We will now describe all these four steps in more detail.

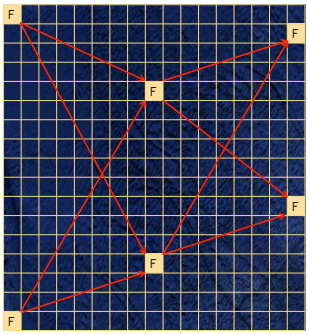

Figure 6: Hypothesis Generation

4.1 Hypothesis Generation

Considering a grid of $n \times n$, the initial lawnmower identifies flat cells in column 0, $n/2$, and $n-1$. Let us call these three sets of cells $\mathcal{C}_0$, $\mathcal{C}_1$ and $\mathcal{C}_2$. We generate hypothetical clear segments by connecting every flat cell in $\mathcal{C}_0$ with every flat cell in $\mathcal{C}_1$ and then the same for cells in $\mathcal{C}_1$ and $\mathcal{C}_2$ (see red arrows in Figure 6). We refer to these segments as our *hypotheses*.

Given these hypotheses with their corresponding initial and final cells, we also store information about other cells in the grid (shown in orange colour in Figure 7), which will be useful to validate or discard hypotheses in the next steps of the procedure. In particular, for each hypothesis h_i, we identify the cells that contain the intersection points between h_i and each other hypothesis h_j (see, for example, cells 7 and 15 in Figure 7) and the cells that contain the middle points between the initial or final cell of h_i and the intersection cell (see, for example, cells 3, 4, 8 and 9 in Figure 7). Finally, if the hypothesis h_i does not intersect other hypotheses, we take the cell that contains its middle point (see, for example, cells 5 and 6 in Figure 7). The intersection and middle cells are interesting because, if the AUV inspects one of such cells and finds complex seabed in it, the hypotheses that pass through that cell can be discarded since they cannot represent valid components of clear passages. In what follows, we identify each cell with its corresponding central point and use the terms *waypoints* and cells interchangeably.

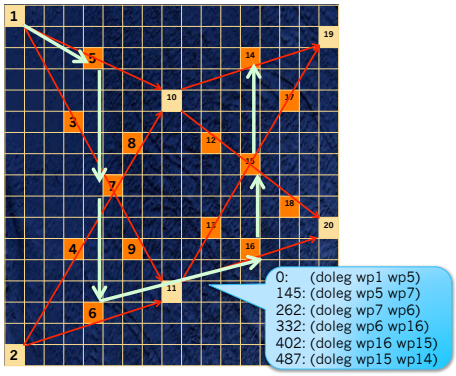

Figure 7: The waypoints of interest (in orange) and the initial plan (green segments and blue box).

4.2 Formulation of Planning Tasks

We use the language PDDL2.2 (Planning Domain Definition Language) (Edelkamp and Hoffmann 2004; Fox and Long 2003) to model the reconnaissance problem, taking advantage of several sophisticated features of this language to express all the properties of our domains.

We model all the relevant waypoints in our problem and their distances. As discussed in Section 3, they are the known flat waypoints identified during the lawnmower, the intersection points between hypotheses and the middle points of the hypotheses. We then associate a *reward* with each waypoint that represents an estimate of how many hypotheses can be disproved if that waypoint is found to be complex. These rewards guide the planner to explore the regions where the highest number of candidate paths can be discarded. By iteratively discarding more and more hypotheses, the search can then be directed towards the most promising regions where a clear passage can be found. The rewards are updated over time so that no hypothesis is checked more than once. A total reward keeps track of the accumulated capacity of the plan to be effective in discarding hypotheses.

The basic structure of the domain for the reconnaissance problem is simple: there is a navigation action that allows the vehicle to move from one waypoint to another. It has an entry waypoint and an exit waypoint. The effect, other than to move the vehicle from the entry to the exit point, is to increase the total reward by a quantity that represents the specific reward associated with the exit point discounted by the distance of such point from the entry point. This way of calculating the total reward allows us to maximise the opportunities of discarding unsafe hypotheses and also to minimise the total distance traversed by the AUV. The actions are durative and their duration is fixed in the problem instance to be the correct (computed) value for the traversal of the distance from the entry to the exit point.

Figure 8 displays the definition of the reconnaissance problem in PDDL2.2.

4.3 Planning Mechanism and Plans

We use POPF-TIF (Piacentini et al. 2015) to build plans for the observer. POPF-TIF is an advanced version of the planner OPTIC (Benton, Coles, and Coles 2012) (Optimizing Preferences and TIme-dependent Costs) and POPF (Coles et al. 2010), that allows us to handle complex temporal interactions. POPF-TIF is a temporal planner for use in problems where the cost function is not directly linked to the plan make-span, as it usually happens, but can be expressed as a continuous function. It combines grounded forward search with linear programming to handle continuous linear numeric change. POPF-TIF performs anytime, cost-improving search: it finds a first solution very quickly, since the empty plan is already a feasible solution, but it then spends the additional time improving on this solution by adding further manoeuvres to the plan or by trying different collections of manoeuvres. The search uses a weighted-$A^\star$ scheme with steadily changing weights in a tiered fashion. The plans produced are monotonically improving, so the final plan is selected for execution. We use a time-bounded search limited to 10 seconds because we are in a time-critical situation (this value is a configurable parameter). We use POPF-TIF because it is very fast at producing its first solution and provides an any-time improvement behaviour.

Figure 7 shows an example of a plan generated by POPF-TIFs for the single vehicle reconnaissance domain. Plans for this domain look like sequences of waypoints to visit. In particular, the planner chooses waypoint sequences that maximise the opportunity to discard unsafe passages and, at the same time, minimise the use of resources. In Figure 7, for example, the planner chooses the path $wp_1 \rightarrow wp_5 \rightarrow wp_7 \rightarrow wp_6 \rightarrow wp_{16} \rightarrow wp_{15} \rightarrow wp_{14}$, which is the optimal route to inspect all the hypotheses while minimising

```
(define (domain recon)
  (:requirements :typing :durative-actions :fluents :timed-initial-literals :conditional-effects)
  (:types waypoint hypothesis)
  (:predicates (at ?p - waypoint) (notVisited ?p - waypoint))
  (:functions (reward) (rewardof ?w - waypoint) (distance ?p1 ?p2 - waypoint) (mf ?w1 ?w2 - waypoint) )

  (:durative-action doLeg
   :parameters (?from ?to - waypoint)
   :duration (= ?duration (distance ?from ?to))
   :condition (and (at start (at ?from)) (over all (not (= ?from ?to))) (at start (notVisited ?to)))
   :effect (and (at end (at ?to)) (at start (not (at ?from))) (at start (not (notVisited ?to)))
    (at end (increase (reward) ( - (rewardof ?to) (distance ?from ?to))))
    (forall (?w - waypoint) (at end (assign (rewardof ?w) (* (rewardof ?w) (mf ?to ?w))))))
  )
)
```

Figure 8: Navigation action in PDDL2.2

the traversed distance. We call this plan π_1.

4.4 Plan Failure and Replanning

Plans are generated under the assumption that every waypoint that is checked by the AUV is found to be complex and that the corresponding hypotheses can be consequently discarded. However, there might be cases in which the AUV visits a waypoint to find that it is characterised by a flat surface. When this happens, the plan ceases to be valid and is abandoned. At this point, replanning is triggered. A new problem is formulated based on the information acquired during Phase 1, as before, and also during the execution of the plan until its failure. In particular, a new flat waypoint is added to the problem, which is the waypoint that has provoked the failure of the previous plan, and its corresponding new hypotheses. In addition, hypotheses relating to waypoints that have been found complex are removed. To continue with our example, let us assume that during the execution of plan π_1 the AUV finds that the cell corresponding to wp_7 has a flat surface (see Figure 9). At this point, the plan π_1 is abandoned and a new plan π_2 is generated. As it can be seen in Figure 9, the hypothesis from wp_1 to wp_{10} and its corresponding middle point have been removed from the planning task formulation, while the flat wp_7 and its corresponding hypotheses and middle points have been added. It is interesting to notice that the new plan focuses on checking waypoints that are around wp_7 since this now looks like a promising area to find a clear passage. During the problem formulation, waypoints wp_3, wp_4, wp_8 and wp_9 receive an higher rewards than the other points since disproving hypotheses in a promising area is more important than disproving hypotheses in unknown areas. This is why the planner now prioritises these waypoints with respect to others.

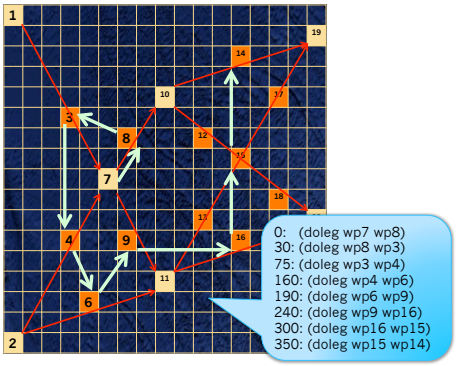

Figure 9: New plan after plan failure.

5 Phase 3: Path Confirmation

Both during Phase 1 and 2, the AUV accumulates information concerning flat cells in the area of operation. At regular intervals, the AUV checks whether the waypoints at the centre of these cells can be connected in such a way to create a path from left to right in the search area. If the AUV finds a potential path and is sufficiently confident that this path might be clear, which depends on how many cells in the path are known to be flat, the AUV exits the planning and replanning cycle and enters Phase 3, i.e. path confirmation. This phase simply consists in the AUV traversing the hypothetical clear path, which is composed of a sequence of segments that connect flat cells between each other. If the AUV confirms that a path has been found, it outputs such a path and the problem is solved. If not, it goes back to Phase 2 and a new task planning problem is formulated that encodes all the new information that has been acquired during the path confirmation phase.

The path confirmation phase is illustrated in Figures 10 and 11. Let us consider the figure on the left first and assume that the AUV already knows that waypoints $wp_1, wp_3, wp_7, wp_9, wp_{11}$ and wp_{20} are all flat. Let us also assume that the AUV now traverses the leg from wp_{11} to wp_{20} and verifies that wp_{16} is flat as well. At this point, starting from wp_{20}, the AUV traverses the path indicated by the green arrows in Figure 11 that goes through $wp_{20} \rightarrow wp_{16} \rightarrow wp_{11} \rightarrow wp_9 \rightarrow wp_7 \rightarrow wp_3 \rightarrow wp_1$. If we assume that the vehicle finds flat in all the cells along this path, then a clear passage is found and the algorithm exits.

6 Implementation

To demonstrate the viability of our approach to underwater reconnaissance, we have developed a simulation in consultation with our industrial collaborators at SeeByte. The

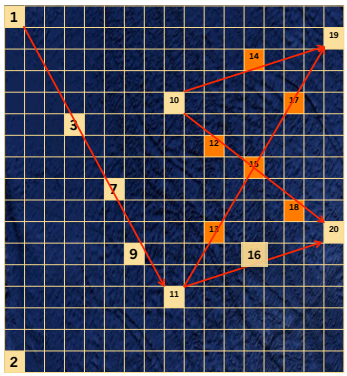

Figure 10: Phase 3: Path Confirmation

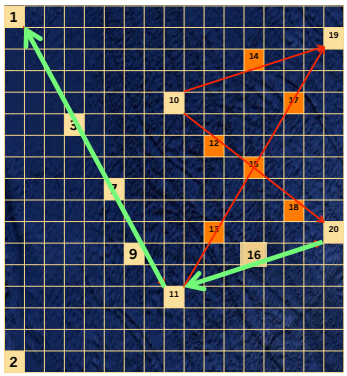

Figure 11: Clear path confirmed

simulation is intended to provide an appropriately abstracted view of the problem. The main abstraction is that we assume the control problem for the underwater vehicles solved. Our implementation can simulate the sea bed and multiple vehicles exploring it and performing reconnaissance tasks. In the next section, we will provide a detailed description of our simulator, while in Section 6.2, we will discuss our initial effort to integrate our plan-based approach to reconnaissance into the SeeByte's proprietary AUV simulator, called SeeTrack, and its relating planning system, called Neptune.

6.1 Architecture of the System

Our simulator, which is based on the DPE (Dreaded Portal Engine) 3D engine, is a cross-platform 3D engine that uses OpenGL 4.4. The AUV is modelled based on the "Girona 500" vehicle and the controller interface uses ROS to communicate with the vehicle. ROS, the Robot Operating System (Quigley et al. 2009), is a set of software libraries and tools used in building robotic systems, which has become increasingly popular both in industry and academia. Sensors, such as sonar and cameras, are simulated in the engine and use ROS topics, services, and actions to send and receive messages. This choice makes it very easy to decouple the simulator from the physical vehicle. Our 3D engine builds on a similar one, which we previously developed and successfully used in the context of the EU project PANDORA (Cashmore et al. 2013) to simulate a multi-AUV environment with sea life.

The general architecture of the system is shown in Figure 12. This is a two-layer architecture, in which the high-level layer (in blue) takes care of the deliberation capabilities of the system and provides an interface towards the bottom-level layer (in green), which corresponds to the control and execution capabilities of the system.

The high-level layer is completely generic and can be reused to embed a generic task planner in any ROS-based execution system. The planner node corresponds to an executable of our planner POPF-TIF (Piacentini et al. 2015) (described in Section 4.3). Although we use POPF-TIF because of its high performance, any other PDDL planner can be embedded in this node. ROSPlan (Cashmore et al. 2015) is at the core of the high-level layer. ROSPlan is a framework

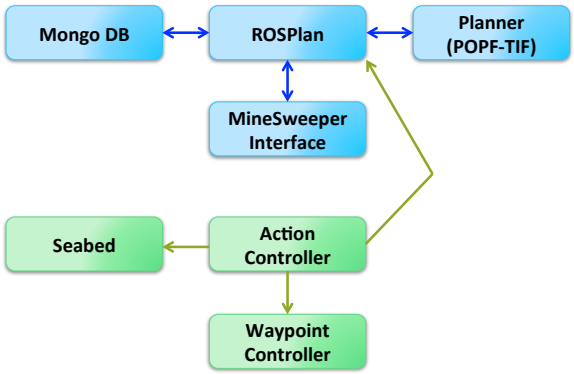

Figure 12: An high-level view of the architecture of our simulation for underwater reconnaissance missions.

for embedding a generic task planner in a ROS system. Together with the planner, ROSPlan is the crucial element of our architecture and provides tools to:

- automatically generate the initial state for the planner from the knowledge parsed from sensor data and stored in a knowledge base;
- automate calls to the planner, then post-process and validate the plan;
- handle the main dispatch loop, taking into account changing environment and action failure;
- match planned actions to ROS action messages for lower level controllers.

Figure 13 presents a general overview of the ROSPlan framework (red box), consisting of the Knowledge Base and Planning System ROS nodes. Sensor data is passed continuously to ROSPlan and is used to construct planning problem instances as well as to inform the dispatch of the plan (blue boxes). Actions are dispatched as ROS actions and executed by lower-level controllers (green box), which respond reactively to immediate events and provide feedback.

Mongo DB implements the knowledge base of our system. It is a collection of interfaces and is intended to collate the up-to-date model of the environment. The knowledge base is updated as soon as new information becomes

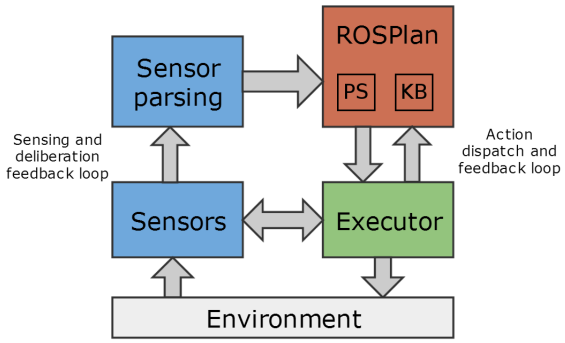

Figure 13: General overview of the ROSPlan framework.

available and is used by the planning system to generate the PDDL problem instance by supplying an initial state and goals. This is done through an interface comprised of ROS services.

Let us now consider the bottom layer. The green modules and the MineSweeper module are application specific and take care of simulating the seabed and one or multiple AUVs exploring such a seabed in the context of reconnaissance missions. The Seabed node represents the model of the seabed and stores the knowledge that the AUVs acquire during their explorations. The model is a grid and, for each cell, we record whether the cell is flat or complex and if an AUV has observed it. As discussed in Sections 3 and 4, we use this model to create the initial state for the planner and update it based on the outcome of the exploration. During execution, the Action Controller node listens to the action dispatch topic of ROSPlan and translates each dispatched action to a command for a controller. It also monitors all the active controllers and provides feedback to ROSPlan. In our case, there is a single controller, the Waypoint Controller node. This controller receives 3D coordinates that are translated by the Action Controller and handles the movement of the AUVs toward the given coordinate during execution. The MineSweeper node sets up ROSPlan for our specific application.

Figures 14 and 15 concern a one vehicle mission and show the vehicle that performs the initial lawnmower and executes the plan initial respectively. In Figure 15, the plan is shown at the bottom and the path followed by the vehicle is highlighted in yellow.

6.2 Integration with SeeByte Neptune

SeeByte, our industrial collaborator, has provided us with details and specifications for modelling problems in the underwater domain and has given us access to their hardware and software products for integrating high-level planning behaviour in them. In particular, SeeByte has developed two main pieces of software:

- SeeTrack: this is the overall infrastructure to directly control and monitor the AUVs. It has a graphical interface that allows the operator to send commands to the vehicle and to receive updates. It contains maps of the area of the operation, one window for each AUV with many parameters to set and observe.
- Neptune: this is the software in charge of autonomy. It has a graphical interface, more sophisticated than SeeTrack, through which a human operator can plan a high-level mission. Planning a mission means specifying the number of vehicles available, the area that needs to be surveyed, the targets that need to be inspected, what polygons to use around the targets to analyse them and what exclusion zones are present in the area of operation, which are regions that the AUVs cannot enter. Upon the description of a mission, Neptune creates a sequence of actions to execute the mission and push it to the AUVs. Neptune supports a limited form of autonomy based on the human operator's input and is domain-dependent (Patrón, Lane, and Petillot 2009).

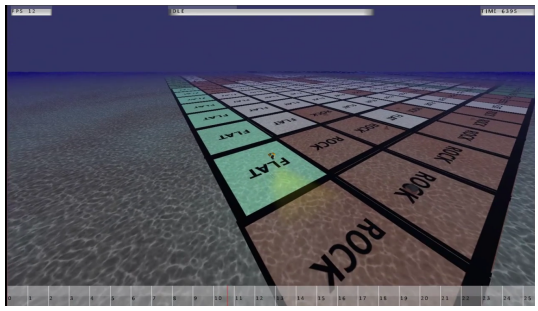

Figure 14: One vehicle performing a lawnmower.

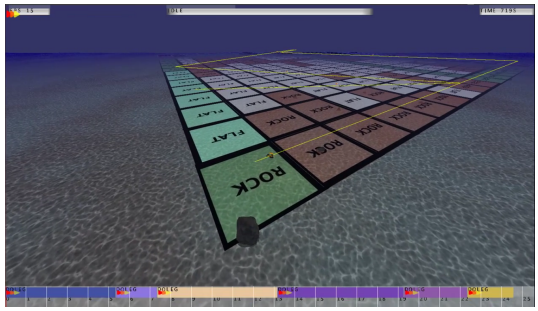

Figure 15: One vehicle executing the plan.

SeeByte has also built a physical AUV simulator. It can simulate up to two unmanned maritime platforms (surface and underwater), sensors detections, communication links, environmental injects, time clocks and temporal events. It also supports combination of real and virtual platforms and sensors. The core Neptune modules are the same as run on real assets. The simulator is very useful as it allows the operator to experiment with a mission in simulation before running the same mission file on real assets.

The long-term goal of our work is to integrate our plan-based reconnaissance behaviour in the SeeByte AUV simulator, which involves linking ROSPlan with Neptune. The integration between Neptune and ROSPlan is facilitated by

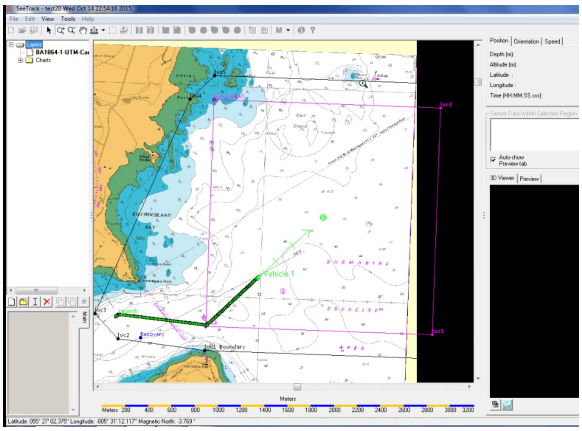

Figure 16: Screenshot of SeeTrack with a vehicle performing a reconnaissance mission. Commands are sent from ROSPlan to the vehicle simulator directly.

the use of ROS in both systems. So far, we have demonstrated that is possible to connect ROSPlan with the SeeByte vehicle simulator by directly dispatching actions from ROSPlan to the vehicle controllers (see screenshot in Figure 16). The next step is to create an interface between ROSPlan and Neptune. Although this is feasible, there are engineering challenges involved in this since both Neptune and ROSPlan have been built to handle the dispatch loop. In order to function together, a hierarchy of command needs to be established that will decide when one must cede to the other. This will require that the protocol between them for determining precedence and the points at which control is devolved must be defined, which will require technical input from SeeByte.

7 Next Steps

We have already extended our work to deal with reconnaissance missions that employ multiple heterogenous assets that need to cooperate with each other during the search for a path. A search plan is devised centrally, decomposed into the sub-plans for individual vehicles and distributed to them for execution. Assuming that these vehicles can only communicate on the surface, the plan contains a number of communication activities which require all AUVs to surface in the same time windows in order to share information about the developing path. However, poor communications can lead to plan failure during plan execution, which force replanning by individual AUVs. We have devised a method for resolving failures within the sub-plans in such a way that, when replanning, the single AUVs do not violate the planned commitments that they have already made with the other assets to cooperate and to communicate during the task at hand.

In future work, we intend to perform a broad experimentation of our plan-based approach, both in simulation and in real sea trials. We aim to complete the integration between our planning system with Neptune, with the goal of managing command-and-control loop for plan dispatch and hand-over between plan execution and replanning phases.

In addition, we will extend initial work on sea-bed models to allow exploitation of probabilistic initial state construction in the deterministic planning models and explore the problem of deployment on larger fleets of cooperating assets across multiple modes (UAVs, USVs and AUVs).

References

Benton, J.; Coles, A.; and Coles, A. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In *Proceedings of the Twenty Second International Conference on Automated Planning and Scheduling (ICAPS-12)*.

Bernardini, S.; Fox, M.; Long, D.; and Bookless, J. 2013. Autonomous Search and Tracking via Temporal Planning. In *Proceedings of the 23st International Conference on Automated Planning and Scheduling (ICAPS-13)*.

Bernardini, S.; Fox, M.; Long, D.; and Piacentini, C. 2016. Leveraging Probabilistic Reasoning in Deterministic Planning for Large-Scale Autonomous Search-and-Tracking. In *Proceedings of the 26st International Conference on Automated Planning and Scheduling (ICAPS-16)*.

Bernardini, S.; Fox, M.; and Long, D. 2014. Planning the Behaviour of Low-Cost Quadcopters for Surveillance Missions. In *Proceedings of the 24st International Conference on Automated Planning and Scheduling (ICAPS-14)*.

Bernardini, S.; Fox, M.; and Long, D. 2015. Combining Temporal Planning with Probabilistic Reasoning for Autonomous Surveillance Missions. *Autonomous Robots* 1–23.

Cashmore, M.; Fox, M.; Larkworthy, T.; Long, D.; and Magazzeni, D. 2013. Planning inspection tasks for auvs. In *In Proceedings of OCEANS'13 MTS/IEEE*.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. ROSPlan: Planning in the Robot Operating System. In *In Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS-15)*.

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

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. In *Proceedings of the 4th International Planning Competition (IPC-04)*.

Faria, M.; Pinto, J.; Py, F.; Fortuna, J.; Dias, H.; Martins, R.; Leira, F.; Johansen, T. A.; Sousa, J.; and Rajan, K. 2014. Coordinating uavs and auvs for oceanographic field experiments: Challenges and lessons learned. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 6606–6611.

Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research* 20.

Patrón, P.; Lane, D. M.; and Petillot, Y. R. 2009. Continuous mission plan adaptation for autonomous vehicles: balancing effort and reward. In *4th Workshop on Planning and Plan*

Execution for Real-World Systems, 19th International Conference on Automated Planning and Scheduling (ICAPS'09), 50–57.

Piacentini, C.; Alimisis, V.; Fox, M.; and Long, D. 2015. An extension of metric temporal planning with application to AC voltage control. *Artificial Intelligence* 229:210–245.

Quigley, M.; Gerkey, B.; Conley, K.; Faust, J.; Foote, T.; Leibs, J.; Berger, E.; Wheeler, R.; and Ng, A. 2009. ROS: an open-source Robot Operating System. In *Proceedings of the International Conference on Robotics and Automation (IJCAI)*.

SeeByte. 2013. Seetrack neptune - user manual.