T-REX: A Model-Based Architecture for AUV Control

Conor McGann, Frederic Py, Kanna Rajan, Hans Thomas, Richard Henthorn, Rob McEwen
Monterey Bay Aquarium Research Institute, Moss Landing, California
{cmcgann,fpy,kanna.rajan,hthomas,henthorn,rob}@mbari.org

Abstract

Autonomous Underwater Vehicles (AUVs) are an increasingly important tool for oceanographic research demonstrating their capabilities to sample the water column in depths far beyond what humans are capable of visiting, and doing so routinely and cost-effectively. However, control of these platforms has relied on fixed sequences for execution of pre-planned actions limiting their effectiveness for measuring dynamic and episodic ocean phenomenon. In this paper we present an agent architecture developed to overcome this limitation through on-board planning using Constraint-based Reasoning approaches. Preliminary versions of the architecture have been integrated and tested in simulation and at sea.

Introduction

Oceanography has traditionally relied on ship-based observations. These have recently been augmented by robotic platforms such as Autonomous Underwater Vehicles (AUV) [1-4], which are untethered powered mobile robots able to carry a range of payloads efficiently over large distances in the deep ocean. A common design relies on a modular tube-like structure with propulsion at the stern and various sensors, computers and batteries taking up the bulk of the tube (Fig 1). AUVs have demonstrated their utility in oceanographic research in gathering time series data by repeated water-column surveys [5,8], detailed bathymetric maps of the ocean floor in areas of tectonic activity [6] and performed hazardous under-ice missions [7].

Typically AUVs do not communicate with the support ship or shore while submerged and rely on limited stored battery packs while operating continuously for tens of hours. In addition to operating without human intervention, AUV operations are exacerbated by uncertainty in the environment, the platform and the mission. Environmental uncertainty (white noise, bio-fouling of sensors, lack of visible light beyond the upper 150m of the photic zone, hazards due to uncharted underwater terrain and ocean currents etc) and Platform uncertainties (sensor failures, battery failures, impaired mobility due to entanglement in fishing lines etc) all add to the difficulty of operating in this harsh environment. Dealing with scientific uncertainty (to observe dynamic and episodic phenomenon) has additionally become paramount for ocean scientists to enable understanding large-scale ecological process.

Fig 1. An MBARI AUV at sea

Current AUV control systems [9] are a variant of the behavior-based Subsumption architecture [23]. A behavior is a modular encapsulation of a specific control task and includes acquisition of a GPS fix, descent to a target depth, drive to a given waypoint, enforcement of a mission depth envelope etc. An operator defines each plan as a collection of behaviors, which are scripted a priori using simple mission planning tools. Behaviors include specific parameters for start and end times as well as a maximum duration. In practice, missions predominantly consist of sequential behaviors with duration and task specific parameters equivalent to a linear plan with limited flexibility in task duration. Such an approach becomes less effective as mission uncertainty increases. Further, the architecture offers no support to manage the potentially complex interactions that may result amongst behaviors, pushing a greater cognitive burden on behavior developers and mission planners. This paper describes initial steps in demonstrating onboard automated planning to generate robust mission plans using system state and desired goals. We expect this approach to reduce the cognitive burden on AUV operators since missions will now be specified as high-level goals and constraints rather than detailed behavior sequences. Our interest in the near term is to incorporate a decision-making capability onboard

AUVs to make this platform more adaptive in the face of uncertainty.

The remainder of this paper is laid out as follows. We start with an overview of our agent architecture followed by preliminary results and related work in this domain. We close with conclusions and future work.

The T-REX Architecture

T-REX (Teleo-Reactive EXecutive) is a goal-oriented system, with embedded automated planning [24,25] and adaptive execution at its core. The system encapsulates the long-standing notion of a *sense-deliberate-act* cycle at the heart of a control loop and reflects the goal-oriented nature of control and provision of response to exogenous events. In order to make embedded planning scalable the system enables the scope of deliberation to be partitioned functionally and temporally and to ensure the current state of the agent is kept consistent and complete during execution. While T-REX was built for a specific underwater robotics application, the principles behind its design are applicable in any domain

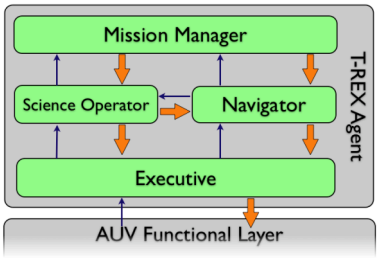

Fig. 2: A 4-reactor T-REX agent.

where deliberation and execution are intertwined.

Fig. 2 shows a conceptual view of a Teleo-Reactive *Agent*. An agent is viewed as the *coordinator* of a set of concurrent control loops. Each control loop is embodied in a Teleo-Reactor (or reactor for short) that encapsulates all details of how to accomplish its control objectives. Arrows represent a messaging protocol for exchanging facts and goals between reactors: thin arrows represent *observations* of current state; thick arrows represent *goals* to be accomplished.

Reactors are differentiated in 3 ways:

- Functional scope: indicating the state variables of concern for deliberation and action.
- Temporal scope: indicating the look-ahead window over which to deliberate.
- Timing requirements: the latency within which this component must deliberate for goals in its planning horizon.

Fig. 2 for example, shows four different reactors:

- The *Mission Manager* provides high-level directives to satisfy the scientific and operational goals of the mission: its temporal scope is the whole mission, taking minutes to deliberate if necessary.
- The *Navigator* and *Science Operator* manage the execution of sub-goals generated by the *Mission Manager*. The temporal scope for both is in the order of a minute even as they differ in their functional scope. Each refines high-level directives into executable commands depending on current system state. The *Science Operator* is able to provide local directives to the *Navigator*. For example if it detects an ocean front it can request the navigation mode to switch from a Yo-Yo pattern in the vertical plane to a Zig-Zag pattern in the horizontal plane, to have better coverage of the area. Deliberation may safely occur at a latency of 1 second for these reactors.
- The *Executive* provides an interface to a modified version of the existing AUV functional layer. It encapsulates access to commands and vehicle state variables. The *Executive* is reasonably approximated as having zero latency within the timing model of our application since it will accomplish a goal received with no measurable delay, or not at all; in other words it does not deliberate.

T-REX has a central and explicit notion of time with all reactors synchronized by an internal clock. The unit of time is a *tick*, defined in external units on a per application basis; tick boundaries signify when synchronization of all reactors must occur while between ticks reactors may deliberate. The agent-state is represented as a set of timelines, which capture the evolution of a system state-variable over time. A timeline is a sequence of *tokens* that are temporally qualified assertions. An assertion is expressed as a predicate with start and end time bounds defining the temporal scope over which it holds. The minimum duration of a token is a tick giving a discrete synchronous view of the state of the world. Token start and end times can be defined as intervals to express temporal flexibility.

Agent timelines are distributed across reactors depending on their functional scope. Information exchange between reactors, where necessary, is provided through the following mechanisms:

- Explicit timeline ownership: Each timeline is owned by exactly one reactor. Any reactor may request a new goal, or replan such requests in the

event of a change of plan; but only the owner of the timeline can decide what goal to instantiate.

- Observations: capture the current value of a timeline. Observations are asserted by the owner of a timeline.
- Goals: express a desired future timeline value. They offer a way to delegate a task to a reactor. Goals are requested for expansion into sub-goals or commands and can be recalled on plan changes when replanning is triggered.
- Dispatch and notification rules: define when information must be shared to ensure consistency and completeness of agent state at the execution frontier and to allow sufficient time for deliberation.

The mapping between reactors and timelines is the basis for sharing information. If a reactor owns a timeline it is declared *internal* to that reactor. If a reactor uses a timeline to observe values and/or express requirements it is declared *external* to that reactor. Fig. 3 illustrates the flow of information in a system containing 3 reactors: The *Mission Manager* keeps track of science goals to give directives to the *Navigator* using the *Path* external timeline. The *Navigator* manages the navigation of the AUV with one *internal* timeline and three external timelines. The navigation route is used to select the appropriate commands to send to the *Executive* as an internal timeline while Position and Attitude timelines capture AUV navigation data. A Command timeline captures the command state of the *Executive*. These external timelines are *internal* to the *Executive* in turn. The Command timeline values are the actual commands that are managed by the AUV functional layer. The content of this last timeline at the execution frontier corresponds to the currently active behavior.

To ensure a complete and consistent view of system state, the T-REX information exchange framework needs to impose further restrictions on the way timelines, observations and goals can be used:

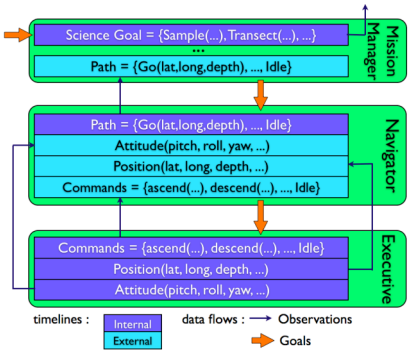

Fig 3: Flow of information between timelines

- No 'holes' are allowed at the execution frontier i.e. all timelines must have a value at the end of the current tick.
- If no update is provided via an observation, and in the absence of information to the contrary, a reactor assumes the previous value(s) on the timeline is/are still valid. We refer to this as the *Inertial Value Assumption* since it conveys some inherent inertia of current values. Contradictory information can come from the model or from a new observation. This has important implications for reducing the cost of synchronization since observations need *only be published as timeline values change*.
- At the end of the current tick, all observations must be consistent, by requiring all reactors to hold the same view at the execution frontier.
- The past is monotonic. All tokens that have finished (i.e. in the past) or that have started but have yet to finish (i.e. they span the execution frontier) can only be restricted in time.
- An observation received at a tick applies to that tick only. It cannot refer to the past except by restricting the values of a token that is actually running (i.e. with an end time in the future). It cannot refer to the future, as it would then be a goal, rather than observed reality.

```
handleTick(tick){
  synchronize (tick);
  dispatchGoals(tick);
  done = false;
  while(!done && currentTick() == tick)
     done = stepNextReactor(tick);
}
```

Fig. 4: The T-REX agent algorithm

The algorithm at the heart of a T-REX agent in Fig. 4 is called at the start of every clock tick. There are three key steps in the algorithm; first, all timelines are synchronized at the current execution frontier. Second, all goals are dispatched. And finally, the remaining CPU time can be allocated to reactors for deliberation in incremental steps. Each of these component algorithms operates over the entire set of reactors.

Synchronization

The goal of synchronization is to produce a consistent and complete view of agent state at the execution frontier. All reactors synchronize at the same rate – once per tick. While this may seem onerous, the actual cost of synchronization is based on how much information has actually changed. For example in Fig 3. the Position timeline is

relatively volatile and will likely change on every tick. However, the Path timeline may hold a single value for many ticks. In this case, as a result of the *Inertial Value Assumption*, if no new observation is received, the Path timeline will extend its current value simply by incrementing the lower bound of the end time of the current value.

The strict rules of timeline ownership enable a clear policy for conflict resolution: observations dominate expectations. For example, if the *Navigator* expected the vehicle depth to be less than 0.3m in order in order to obtain a GPS fix but the actual depth observed by the *Executive* is 1 meter, then the expected value is discarded. This may impact plan feasibility and force the *Navigator* to find an alternative solution by rejecting the current plan.

To ensure global consistency the agent undertakes local synchronization of the reactors until quiescence. In principle, this operation is equivalent to solving a planning problem over the set of all *internal* timelines for a planning horizon restricted to a tick. If a reactor has an external timeline, it depends on its owner for such consistency. In this way the reactors form a dependency graph which in practice we require to be acyclic, allowing ordering of synchronization for purposes of efficiency.

Dispatching Goals

Where observations are the driver for reaction, goals are the driver for deliberation. The purpose of dispatching is to task reactors with new goals in a timely manner. To accomplish this, T-REX provides explicit parameters and rules to govern dispatching.

- λ - The latency of the reactor defined as the worst-case number of ticks to deliberate over a request.
- π - The planning horizon of the reactor quantifying the look-ahead for deliberation.
- τ - The execution frontier expresses the current tick and is a boundary between the past and the future.

To understand the implications of the above parameters, consider the example given in Fig. 5. To satisfy the goal *Go*(31.73, -121.80, 100) in its *Path* timeline the *Navigator* decides that it needs the vehicle to *descend*(100) at tick 10 for a duration between 50 and 55 ticks and then to achieve *waypoint*(31.73,-121.80) on successful termination of *descend*. Since the *Executive* is the owner of the *Command* timeline, these two goals need to be dispatched by the *Navigator* to the *Executive* so that the latter can resolve them. The importance of λ is to ensure the *Executive* has

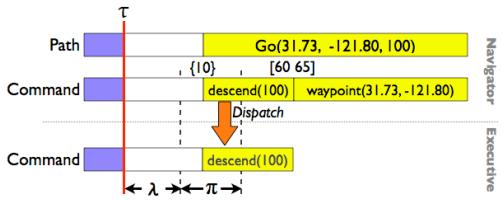

Fig. 5: Illustration of goal dispatching window

sufficient time to complete deliberation prior to starting the requested goal. If the start-time for a goal dispatched to the *Executive* at τ were necessarily less than $\tau+\lambda_{Exec}$ the *Executive* may be unable to deliberate to resolve the goal, leading to a plan failure.

Since the planning window of the *Executive* is π_{Exec}, the Executive should receive all goals that can start before $\tau+\lambda_{Exec}+\pi_{Exec}$. This will enable the *Executive* to leverage as much information as it can handle in making judicious decisions on how to accomplish the goals requested. Sending a goal with a start time *strictly greater* than $\tau+\lambda_{Exec}+\pi_{Exec}$ will not be considered by the *Executive*. Moreover, such dispatch incurs a cost for transmission of information and may over-commit the *Mission Manager* unnecessarily.

Therefore the general rule is that the *dispatching window for a timeline is a time window that depends on the latency and the look ahead of the reactor owning the timeline*. This dispatch window, H_D is defined by the following:

$$H_D = [\tau + \lambda, \tau + \lambda + \pi]$$

This implies that as soon as the start time of a goal on an external timeline intersects H_D, it is dispatched to the owner of the timeline. This rule is necessary and sufficient to ensure that each reactor has sufficient time (λ) and information (π) to deliberate on goals provided by other reactors. In our implementation, we have an *Executive*, which is purely reactive and therefore $\lambda_{Exec} = \pi_{Exec} = 0$ implying that the *Executive* does not plan beyond the execution frontier.

Deliberation

The framework presented thus far makes the details of deliberation an internal concern for each reactor even if it has to capture different functional and temporal scope. Our own implementation of T-REX uses a Constraint-based Temporal Planning approach based on EUROPA-2 [24,25].

Deliberation employs a declarative model-based paradigm. The model describes state variables (e.g.

```
class Path extends AgentTimeline {
  predicate At{Node location;}
  predicate Go{Node from; Node To;}
}
class Position extends AgentTimeline {
   predicate Holds{Node value};
}
Path::At {
   met_by(Go g);
   eq(g.to, location);
   contained_by(Position.Holds p);
   eq(p.value, location);
}
Path::Go {
  met_by(At p);
  eq(p.location, from);}
```

Fig. 6: Example domain model in EUROPA

position, battery level) and actions (e.g. *ascend*, *descend*, *getGPS*, *takeWaterSample*) of the system. Constraints can be specified to enforce relationships between state variables. For example, it is convenient to represent the vehicle as being at the surface, or not, which can be captured with a boolean state variable (e.g *AtSurface*). We define a relationship between this variable and the deph of the vehicle as follows: if *depth* <= 0.3 then *AtSurface* = true. The model also describes constraints between states and actions. For example, the vehicle must be at the surface during *getGPS*. A sample domain model is shown in Fig. 6 with the Path timeline having two predicates *At* and *Go*; the example rules in the parameter specification express the constraint that to be at a location, the AUV needs to go to that coordinate and the position must be maintained for a temporal interval that is consistent with the rest of the model. A T-REX agent uses a single model for control at various levels of abstraction and at various speeds of execution. Different reactors reference subsets of this model according to their functional scope.

The Deliberative reactor is a specialization of a Teleo-Reactor utilizing models, plans and planning to accomplish reactive and goal directed control. Fig. 7 describes the main components of this reactor. The inward pointing arrows reflect the invocations of the agent control loop for synchronization, dispatch and deliberation. The Database is a source and sink for observations and goals based on the semantics of internal and external timelines and the rules of information exchange. It is an extension of the EUROPA-2 plan database, augmented for specialized buffering for efficient access to timeline data for dispatch and synchronization and manages state information. Model rules are applied automatically through a combination of propositional inference and constraint propagation [32], to check consistency and prune infeasible elaborations of the plan maintained in the database. The Synchronizer is a specialized configuration of a EUROPA solver operating over a 1-tick horizon. It accomplishes local consistency and completeness. The database propagates the results of synchronization to the future. The Dispatcher is a simple algorithm that publishes goals to owner reactors of its external timelines according to the dispatch semantics previously defined. Finally, the Planner is yet another instance of a EUROPA solver used to deliberate over the specified temporal and functional scope of the reactor using a heuristic based chronological backtracking search for partial plan refinement. These entities together are used under different configurations for the Mission Manager, Science Operator and Navigator shown in the example in Fig 3. Details on EUROPA can be found in [20,21].

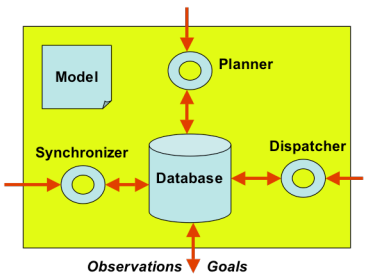

Fig. 7: A Deliberative reactor

Experimental Results

Our AUV uses two onboard computers: a main vehicle computer which is a 244 Mhz PC/104 stack running the QNX real-time operating system, and a separate 367 MHz EPIC EPX-GX500 AMD Geode stack running Linux and T-REX. The communication between the *Executive* and the functional level computer is a socket-based protocol allowing the exchange of command requests (i.e. goals) and state updates (i.e. observations). For validation purposes we first ran experiments on a high-fidelity AUV simulator based on [23]. This simulator captures vehicle dynamics that were used to validate our missions prior to going to sea.

At the time of writing we have completed two sea

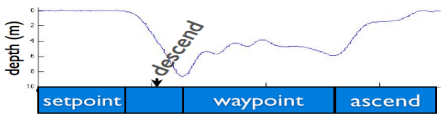

Fig. 8: A simple plan execution on AUV.

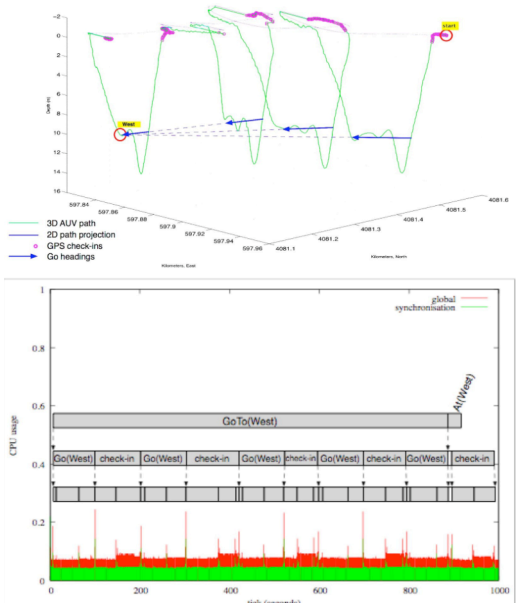

Fig. 9 A navigation mission (top) with its T-REX plan and CPU profile (bottom)

trials: the first was to validate basic system performance under real conditions and the second to work on higher level planning (*Mission Manager*) interleaved with the execution control (*Navigator* and *Executive*). These sea trials were in the northern Monterey Bay using our support ship the R/V Zephyr. Fixing the tick duration to 1 second is adequate for our application.

Initial runs were to demonstrate nominal sequencing primitives such as *setpoint(v)* to get the vehicle moving, *descend(d)*, *waypoint(lat, long)* to go at a given position and *ascend(0)*. The role of T-REX in these runs was limited to checking sequence validity and to track execution. Fig. 8 shows a sequence execution for a 400 second run and a CPU usage around 7%, primarily for

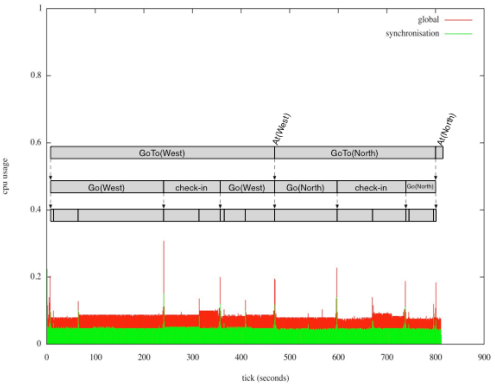

Fig. 10 A multi-node mission

synchronization.

After this initial validation and during the second trial we tested the interleaving of reactive planning and execution. Fig. 9 shows a plan given to the *Navigator* with the goal to be at the *West* node at 10m depth and running for a duration of 1000 seconds. The *Navigator* generated a plan to dynamically change the AUV's attitude while heading towards *West* node and inserting periodic *check-in* windows to allow localization at the surface using GPS. During execution we observed an average CPU usage of 8.5% with peaks at end of actions consistent with synchronization and plan adaptation. The larger peaks (25% of CPU usage) correspond to situations implying deliberation at a higher level. For example these peaks are generally at the end of the abstract actions such as *Go* and *Check-in* that triggered deliberation in the *Navigator*.

Fig.10 shows the execution of a more complex mission where the *Mission Manager* was able to order the visitation by making the AUV to go first to the *West* and then to the *North* node with the insertion of ordering constraints. This plan was then executed reactively by the *Navigator*, which inserted check-in windows while keeping CPU usage to 8.8%. With an additional reactor adding to the deliberation, this experiment showed that T-REX is capable of managing multiple goals without impacting system responsiveness.

We are now working on higher-level goals connected to science objectives such as "do a transect from A to B in a depth range from 10m to 100m" or "track and characterize an ocean Front" allowing scientists to specify abstract mission goals.

Related Work

T-REX's legacy is derived primarily from the Remote Agent Experiment or RAX [24,25]. Further T-REX is similar to IDEA [26,27] in its formulation of a timeline-based representation, and in its use of planning at the execution frontier and for deliberation. It is distinct from IDEA primarily in its formulation for exchanging and synchronizing state between reactors. The Autonomous Sciencecraft Experiment [28] conceptually follows the same path as RAX with CASPER a planner that is both deliberative and reactive. CASPER is an adjunct to a separate executive rather than directly embedded in the execution loop as is the case with T-REX. Furthermore, temporal flexibility in T-REX provides greater robustness to temporal uncertainty in comparision to CASPER's grounded representation. The LAAS architecture [29,30]

provides decisional capabilities using a constraint-based symbolic planner integrated with reactive components for autonomy. It is a 3-layered architecture where all the different components (functional modules, execution controller, functional executive and planner) are manipulating different kind of formalisms specified on heterogeneous modeling languages. Such an approach tends to make platform design and integration difficult [31] and prone to errors especially in model design. In contrast, although T-REX does result in a factoring of computation into layers in practice, a hierarchical structure is not inherent in the design, nor is deliberation required or prohibited for any layer.

While a number of deliberative control architectures have been built for AUV control [10-18] T-REX's design philosophy is closest to ITOCA [11], DAMN [16] and ORCA [18]. Both DAMN and ITOCA are based on a reactive Subsumption based architecture with no inherent deliberation. ORCA uses p-schemas in a case-based planning framework; however the efficacy of ORCA's approach is unclear in terms of scalability in the number of schemas nor does it reason explicitly about time and resources. The literature moreover does not indicate whether any field trials were conducted with ORCA on an AUV for validation of this approach.

Conclusions and Future Work

In this paper, we have shown a design of an agent architecture that demonstrates onboard planning and execution to enable the next phase of scientific research in the ocean sciences. While results shown are preliminary, our high-fidelity simulation test-bed along with first tests in the open ocean has validated central notions of timing within this control architecture. Our final end of year deployment will include a full-day scientific mission in the Monterey Bay, with the use of the traditional suite of instruments on our AUV. This mission will demonstrate goal-oriented commanding, onboard resource management, geometric 2D path planning, responsiveness to opportunistic science events with re-planning in-situ and graceful recovery from failures injected for demonstration. Our year-end goals are to demonstrate a realistic science scenario in Monterey Bay with the use of an online opportunistic learning [33] triggered by T-REX.

Acknowledgements

This research was supported by the David and Lucile Packard Foundation. We wish to thank NASA Ames Research Center for making the EUROPA planner available.

References

1. Yuh, J., editor, Underwater Robotic Vehicles: Design and Control, TSI Press, Albuquerque New Mexico, 1995.
2. Yuh, J., "Design and Control of Autonomous Underwater Robots: A Survey" Autonomous Robots, 2000 8, 7–24, 2000.
3. Blidberg D.R., "Autonomous Underwater Vehicles: A Tool for the Ocean," Unmanned Systems, vol. 9, pp. 10-15, Spring 1991
4. Bellingham J. G., et.al. "A Second Generation Survey AUV" In IEEE Conference on Autonomous Underwater Vehicles, Cambridge, MA, USA, 1994.
5. Ryan J.P., et.al "Coastal ocean physics and red tides, an example from monterey bay, California". Oceanography, 18:246–255, 2005.
6. Thomas, H., et.al, "Mapping AUV Survey of Axial Seamount", Eos Trans. AGU, 87(52), Fall Meet. Suppl., Abstract V23B-0615, 2006
7. Bellingham, J. G., et.al 2002 "Field Results for an Arctic AUV Designed for Characterizing Circulation and Ice Thickness" AGU, Fall Meeting 2002.
8. Ryan, J.P., et.al "Physical-biological coupling in Monterey Bay, California: topographic influences on phytoplankton", Marine Ecology Progress Series, Vol 287: 28-32 2005.
9. Belingham J. G., J. J. Leonard, "Task configuration with layered control," in IARP 2nd Wkshp on Mobile Robots for Subsea Environments, Monterey, CA May 1994.
10. Barnett, D., et.al "Architecture of the Texas A&M Autonomous Underwater Vehicle Controller". in Proc. of the Symp. on Autonomous Underwater Vehicles Technology, pp. 231-237, 1996.
11. Ridao, P., et.al "On AUV Control Architecture", Proc. International Conference on Robots and Systems, Nagoya, Japan, 2000.
12. Carreras, M., et.al "An overview on behavior-based methods for AUV control", MCMC2000, 5th IFAC Conference, Denmark, 2000
13. Batlle, J. et al., "URIS: Underwater Robotic Intelligent System", Automation for the Maritime Industries, Chapter 11, pp: 177-203, November 2004
14. Batlle, J., P. Ridao, M. Carreras, "An Underwater Autonomous Agent. From Simulation To Experimentation, in Proc. 9th Mediterranean Conf on Control and Automation, Dubrovnik, 2001.
15. Zheng, X. "Layered Control of a Practical AUV". In Proc. of the IEEE Symp on Autonomous Underwater Vehicles Technology, Washington D.C., pp. 142-147, 1992.
16. Rosenblatt, J., et.al "A behavior-based architecture for autonomous underwater exploration" Intnl. J. of Info Sciences, vol. 145, no. 1-2, 2002,.
17. Williams, S.B., I.J Mahon "Design of an unmanned

underwater vehicle for reef surveying" in Proc IFAC 3rd IFAC Symp. on Mechatronic Systems 2004, Manly, Australia.

18. Turner, R. M., Stevenson, R. A. G. "ORCA: An adaptive, context-sensitive reasoner for controlling AUVs". in Proc 7th Intnl Symp. On Unmanned Untethered Submersible Tech. 1991.
19. Brooks, R. A. "A robust layered control system for a mobile robot" IEEE Journal of Robotics and Automation, RA-2:14–23, 1986.
20. Jonsson, A., P. Morris, N. Muscettola, K Rajan, B Smith "Planning in Interplanetary Space: Theory and Practice" AIPS 2000, Brekenridge, 2000.
21. Frank, J., A. Jonsson, "Constraint-based attributes and interval planning," Journal of Constraints, vol. 8, Oct. 2003.
22. Nilsson, N., "Teleo-Reactive Programs for Agent Control," Journal of Artificial Intelligence Research, 1:139-158, January 1994.
23. Gertler, M., Hagen, G.R. "Standard Equations of Motion for Submarine Simulation" Naval Ship Research and Development Center Report 2510, June 1967.
24. Muscettola N., P. P.Nayak, B Pell, B. Williams. "Remote Agent: To Boldly Go Where No AI System Has Gone Before" Artificial Intelligence 103(1-2):5-48, August 1998.
25. Rajan K., et.al, "Remote Agent: An Autonomous Control System for the New Millennium," Proc. Prestigious Applications of Int. Systems, 14th ECAI, Berlin, 2000
26. Muscettola N., G. A. Dorais, C. Fry, R. Levinson, and C. Plaunt, "IDEA: Planning at the core of autonomous reactive agents", in Proc IWPSS, Houston, October 2002.
27. Finzi, A., F. Ingrand, N. Muscettola, "Model-based Executive Control through Reactive Planning for Autonomous Rovers", IROS 2004, Japan, 2004.
28. Chien, S. et.al, "Using Autonomy Flight Software to Improve Science Return on Earth Observing One", J. of Aerospace Computing, Information Communication. April 2005
29. Ingrand, F., S. Lacroix, S. Lemai-Chenevier, and F. Py, "Decisional Autonomy of Planetary Rovers," to appear in "Journal on Field Robotics", 2007.
30. Ghallab, M. , F. Ingrand, S. Lemai, and F. Py, "Architecture and tools for autonomy in space," in ISAIRAS, Montreal, 2001.
31. D. Bernard, et.al, "Remote Agent Experiment: Final Report," NASA Technical Report, Feb. 2000 available at: http://ic.arc.nasa.gov/projects/remote-agent/DS1-Tech-report.pdf
32. Dechter, R., I. Meiri, J. Perl "Temporal constraint networks", Artificial Intelligence, Volume 49, Issue 1-3, pp 61 – 95, 1991
33. Fox. M, D. Long, F. Py, K. Rajan, J. Ryan "In Situ Analysis for Intelligent Control", Proc. Oceans 2007, Aberdeen, June 2007.