Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 1 (2012) 259–268

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2010

Modeling a hybrid reactive-deliberative architecture towards realizing overall dynamic behavior of an AUV

S. K. Das*, S. N. Shome, S. Nandy, D. Pal

Robotics & Automation, CMERI, M.G.Avenue, Durgapur,WB, PIN-713209, CSIR, INDIA

Abstract

Among all existing computational architecture adopted for controlling the behavior of Autonomous Underwater Vehicles (AUVs), the combined deliberative-reactive methodology is the most effective and significant approach towards behavioral control of the vehicle. Much work has been put into it and is available with literature. However, little work has been done in the scope of modeling the system with a view towards simulating and analyzing the dynamic behavior of the system as governed by the hybrid control architecture. This attempt is quite significant at the design stage, wherein fault-diagnosis can be easily done and rectified for. The aim of this paper is to present such a model for the adopted architecture and simulate the dynamic behavior of the system. Discussion regarding the logical organization and integrity between various modules has been presented, including abstraction between device layer and the controlling sub-systems. Overall dynamic behavior of the system has been realized through a hybrid finite system machine (FSM), thereby exhibiting the essential combination between a continuous reactive layer and discrete event-based deliberative sub-system. The required modeling of FSM and control-subsystems has been done with Stateflow/Simulink from Matlab.

Keywords: AUV, system, hybrid architecture, reactive, deliberative, hybrid model, FSM, modeling, stateflow, simulation

1. Introduction

Autonomous underwater vehicles (AUV) are mobile robotic systems, which can operate underwater at great depths (ranging from 150 to 6000 meters) without any human intervention and connections to the surface control station. As of now, it is a proven technology [1] that can be adopted for several scientific and strategic applications like oceanographic explorations to sub-sea mine diffusing operations. Such critical missions require that the AUVs operate in a robust and reliable manner, which in turn demands flawless control software architecture, consisting of a set of well-coordinated functional software modules interfacing with the various sensors, actuators and associated controllers present in the system.

Software architecture corresponds to a logical and conceptual framework [2], which forms the basis for the logical organization of the existing software components (or modules) that are to be integrated into a single

* Corresponding author. Tel.: +91-343-6510238; fax: +91-343-2546745.
E-mail address: subhro_se@rediffmail.com.

1877-0509
doi:10.1016/j.procs.2010.04.029

functional unit called a system. Among the various methodologies and architectural approaches [3][4], which have been adopted for developing highly competitive and reliable control systems for autonomous robotic systems, hybrid deliberative-reactive approach [5] has been regarded as the most effective one. Such an approach involves the features of both deliberative as well as reactive behavioral control paradigms. Deliberative activities include planning and sequencing of tasks with overall management of various states of the system both at the operational level as well as internal. On the contrary, the reactive layer is attributed with greater response towards the environmental changes. However, extensive modeling from the perspective of system dynamics has been done for simulating the control system of an AUV [6] [7] [8]. Nevertheless, such simulation results fail to reflect the dynamic interactions between the various behaviors and are insufficient for realizing the coordinated functioning of the architectural modules. Since, it is very essential to justify the architectural functionalities at the design stage, therefore, it is necessary to mechanize the same through modeling and simulation before approving it for the final state-of-the-art. In this regard, the underlying objective of this paper remains to realize the logical organization between deliberative and reactive modules, and represent their coordinated functioning through appropriate modeling and simulation. Hybrid modeling technique [9] has been adopted for establishing the proposed architecture. A dynamic state-based controller models the set of goal-driven deliberative modules on the one hand, with a dynamic continuous model based on system-dynamics representing the set of reactive behavioral modules on the other. Thus simulating such a hybrid model may be a good representation of the actual performance of the system.

1.1. System specifications

The proposed AUV is required to maintain an operational sequence as shown in Figure 1.

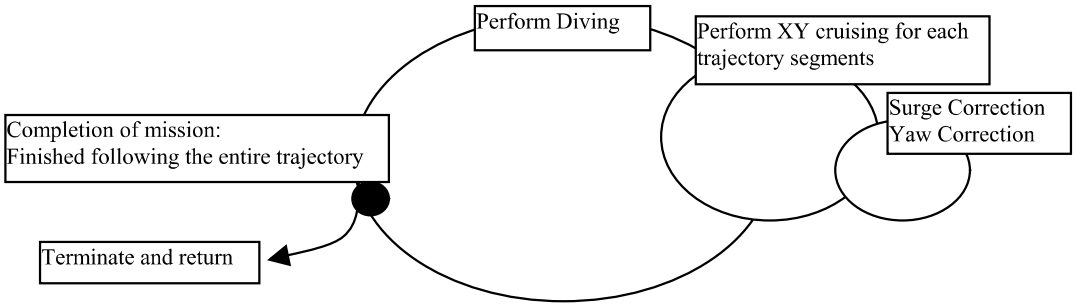


Fig. 1 operational sequence of AUV

As shown in figure 1, the AUV is required to perform diving as its first activity. Only when the desired depth is reached, should it start its cruising operation, i.e. following a given trajectory as a collection of consecutive runs in different directions (heading). In order to simplify the control issues while AUV is in motion the operational requirements of the AUV require that it should follow a given trajectory in piecewise manner. The trajectory is defined as a set of ordered pairs consisting of bearing (i.e. desired heading for the AUV); range (i.e. displacement in the surge direction) as well as hovering time (settling time for stabilizing the system after every segment). The innermost loop is the continuous motion control module, which attempts to achieve the desired surge and heading set points. On the contrary, the middle loop is concerned with updating the reference values for each individual segment of the entire trajectory, which is definitely part of the deliberative activities. The outermost loop is the diving loop, which should execute throughout the mission. Ultimately, when the entire trajectory has been followed the system terminates and the AUV pumps back to surface. It may be clearly observed that the three loops need to be coordinated properly in order to make the system operate successfully, which in turn demands a careful planning and sequencing. It is therefore absolutely proper to state that the required control system needs a controller, which shall decide how the various continuous processes in the system (especially the control loops) should behave based on the occurrence of certain events. This is what essentially constitutes a hybrid system [9]. The physical features of the AUV are presented in Table 1.

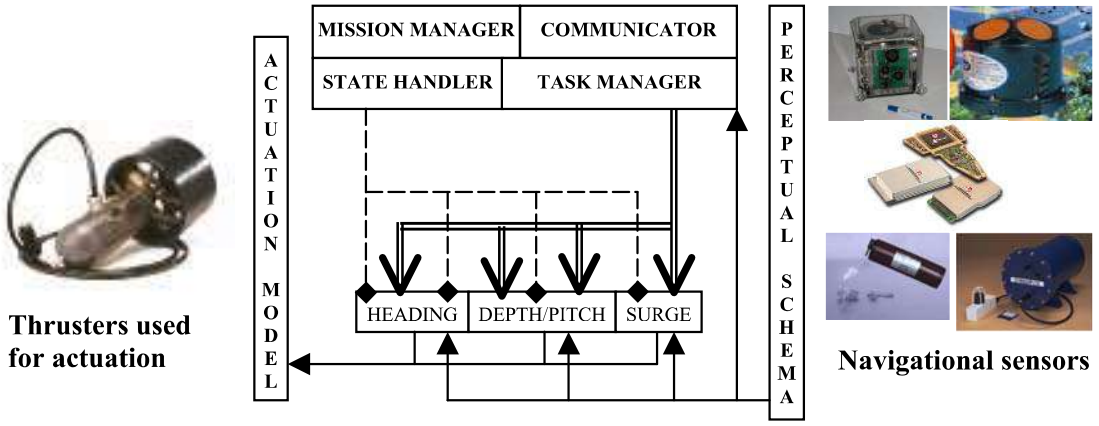
Table 1. Characteristic features of the AUV

Particulars	Description	Particulars	Description
Dimensions / Shape	(4.9x0.5) m/ cylindrical	Actuators	Digital and Analog Thrusters
Mass	490 kg (Air), neutrally buoyant	Navigational Sensors	IMU, DVL, Altimeter, Pressure Sensor
Energy Source	Li Polymer Cells, 6 hrs	Payload	Sonar, CTD, Camera

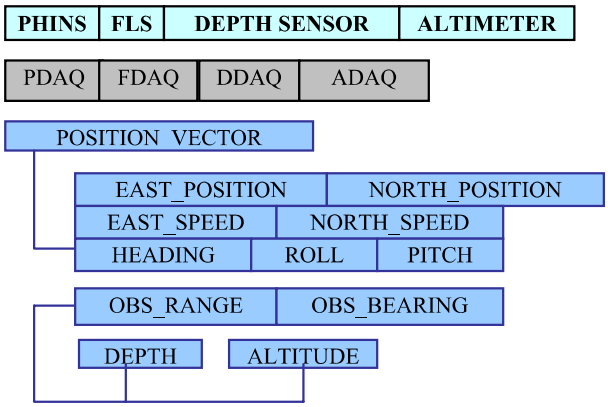
1.2. Overview of the control architecture

A schematic representation of the architectural framework is shown in Figure 2 (a). The overall architecture consists of the deliberative and reactive modules, with two abstraction layers or schemas namely, the actuation model as well as the perceptual schema.

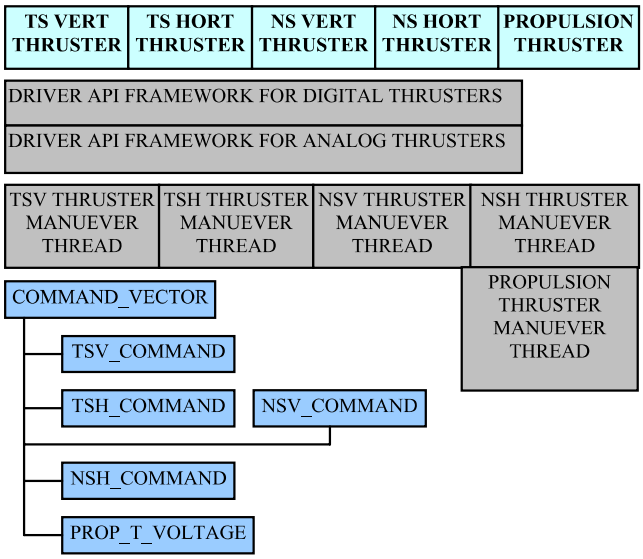
The deliberate modules are responsible for planning, task sequencing as well as governing the dynamic behavior of the system as a whole. As shown in the figure, the broken lines from *State_Handler* to the reactive layer represent the various set points generated by the deliberative module for the continuous reactive control-loops. The *State Handler* is also responsible for deciding the execution status for each of the controller threads and other associated threads executing in the system-process context. The task sequencing and planning is achieved by updating a policy vector as illustrated in Figure 3. The *Task_ Manager* needs to suspend, create and terminate the various threads of execution uses the policy vector thus updated. The policy of controlling execution status of the system is represented in Table 2.



(a)



(b)



(c)

Fig. 2 (a) architecture schematic; details of (b) perceptual schema (c) actuation model

Mission_manager is the entry point to the system context and is responsible for initiating all the other deliberate modules. *Communicator* sub-system is involved with hybrid communication with the surface control, i.e. supporting radio communication while the vehicle is on the surface as well as acoustic communication throughout the period for which the vehicle remains underwater.

Contrastingly, the reactive modules are highly responsive towards environmental changes and work on a greater sampling rate in comparison to the deliberative modules. The reactive layer drives the system towards the desired operating set point, as updated by the goal driven deliberative layer. The continuous control-loops have been modeled and designed on the basis of system dynamics and using conventional control theory.

Sensory feedback from the perceptual schema helps in evolving events, thereby triggering various state transitions inside the *State_Handler*. Table 3 represents the layers comprising both the perceptual schema as well as the actuation model. Both the schemas provide a layer of abstraction to the reactive and deliberative modules, so that the controllers do not have to communicate with the devices directly. Therefore, a change in the device driver or the schemas does not necessitate a modification in the coding for the controller modules. Thus it helps in maintaining and upgrading the architecture even if specifications for the devices (i.e. sensors and actuators) are changed.

Table 2. Policy of task sequencing and controlling execution context of threads

Algorithm for task sequencing (policy_vector)

```
index ← index of policy vector representing a particular thread
index=0
thread ← thread pointed by policy_vector (index)
max_index=TOTAL_NUMBER_OF_THREADS

while (index<=max_index) do
  if (policy_vector (index) == -1)
     if  (thread==running || thread==suspended)
        terminate thread
     else
        Do nothing
     end if
   end if
  if (policy_vector (index) == 0)
     if  (thread==running)
        suspend thread
     else
         Do nothing
     end if
   end if
  if (policy_vector (index) == 1)
     if  (thread==terminated || thread==suspended)
        resume thread
     else if  (thread!=created)
        create thread
     else
        Do nothing
```

end if

end if

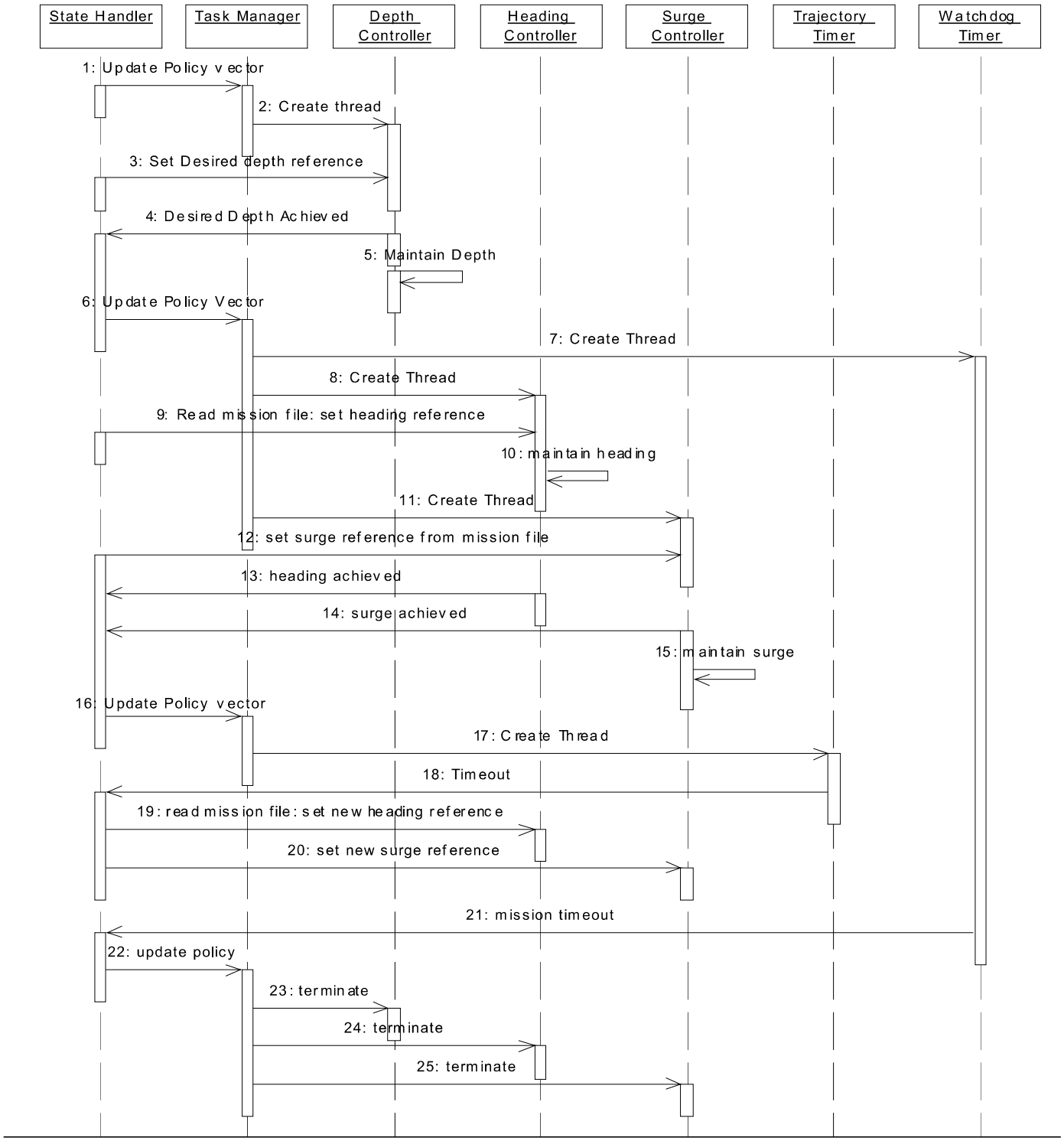


Fig. 3 UML sequence diagram for the overall architecture

1.3. Logical view of architecture

The sequence diagram as shown in figure 3 represents a logical view of the architecture. *State_Handler* is shown to govern the overall execution status of the system through deliberation in two ways, viz. (1) updating the policy vector for thread creation or termination, and (2) updating set points for the reactive controllers. Events are shown to evolve from the reactive modules, when desired depth is reached or a particular segment of trajectory is achieved. The *State_Handler* is shown to acknowledge such events and trigger the trajectory timer to maintain the recently achieved position and hover for a certain period of time unless the timer times out. Whenever the trajectory timer times out it generates an event for the *State_Handler*, on occurrence of which, the *State_Handler* terminates the timer thread and reads the mission file again. Set points for the next segment of trajectory are updated and reactive controller threads shoot off for achieving the same. However, a failure of controllers, improper functioning of thrusters or inconsistency in data received from sensors may lead to a starvation where the vehicle will virtually never reach the desired position. As a result, the mission file will never reach EOF and the mission will continue indefinitely. In order to escape such situations, another emergency timer, i.e. the watchdog timer is used. Whenever the watchdog timer times out it send a signal to the *State_Handler*, which then terminates all the running threads, de-allocates memory, clears device registers and brings the system to a halt (indicated by actions 22, 23, 24 and 25 in figure 3). The dynamic interactions between the *State_Handler* as well as the reactive controller sub-system are verified through hybrid modeling as discussed in the next section.

Table 3. Representative layers of perceptual schema and actuation model

Layers	Perceptual Schema	Actuation Model
Device Layer (uppermost)	Internal sensors like INS, DVL, Depth Log, Altimeter	Analog thrusters for propulsion; Digital thrusters for orientation
Data interfacing layer (middlemost)	Threads contributing to data acquisition	Threads concerned with configuring actuator parameters and generating digital and analog signals for manipulating the thrusters
Data Structure layer ()	Vehicle position; orientation; depth and altimetry	Control signal required to be given to control as well as propulsion thrusters

2. Modeling the system

Functionally, the proposed architecture closely resembles a hybrid system thereby, essentially consisting of two subsystems: (1) a discrete dynamic controller; (2) a set of continuous dynamic processes.

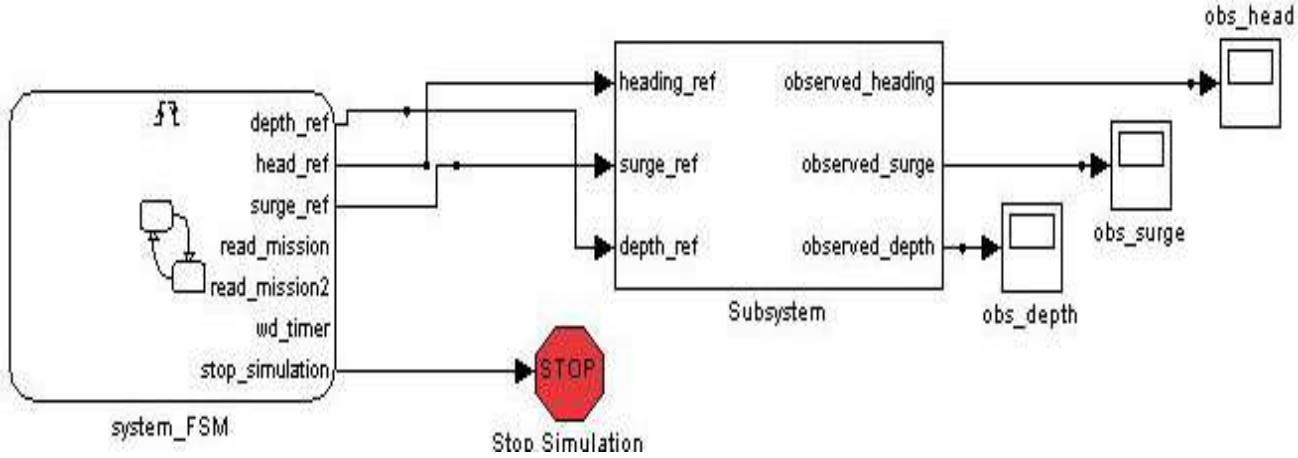


Fig. 4 skeletal framework of the hybrid model

The *State_Handler* aided by the *Task_Manager* as described in the previous section, essentially governs the overall time-varying behavior of the system by coordinating and controlling various threads of execution, which are involved with the reactive layer and operate in a more continuous domain of execution (having lower sampling time). It is therefore, important to develop a model that accurately describes the dynamic behavior of such a system. Only then will it be possible to establish the relations and interaction between the deliberative and reactive components of the architecture, operating in the asynchronous and continuously time-varying domains respectively [10]. For our purpose, we have construed a hybrid model characterized with differential equations as well as FSM based state-machine with the help of StateFlow/Simulink from Matlab. Figure 2 represents a skeletal framework of the model in the simulating environment.

The system_FSM as shown in figure 4, characterizes the *State_Handler* in a reduced form, whereas Subsystem is the model representative for the reactive domain.

2.1. Modeling the reactive subsystem

Since the proposed AUV has only 5 degrees of freedom (with roll balanced by mechanical design), the 6-DOF rigid body equation [11] has been adopted for modeling its transient behavior in the time-domain.

$$M\,(dV/dT)+C(V)\,V+D(V)\,V+g(\eta)=\tau \qquad (1)$$

Where, M is the mass/inertia matrix of the AUV including hydrodynamic added mass/inertia terms, C is the coriolis and centripetal matrix, D is hydrodynamic damping matrix and g is gravity and buoyancy force vector. The hydrodynamic damping is estimated as follows:

$$D = 0.5\,\rho\,A_F V^2 C_D \qquad (2)$$

Where, ρ is seawater density, V is the design velocity of the AUV, A_F is the frontal area and C_D is the viscous pressure drag coefficient. Following considerations are made for required estimation of drag.

Considerations:

Density of sea water, ρ = 1025 kg/m3
Length of the vehicle, L= 4.5m
Maximum diameter, Dmax: 0.5m
Allowance for appendages: 15%
Allowance for roughness: 25%
Frontal area, A_F = 0.1965 square metre for forward motion and 2.0 square metre for cross flow

The plant behaviour during different phases of the mission may be defined as follows:

1. Diving behavior:

$$\tau_z = M_z \dot{\omega} + C_\omega \omega|\omega| + k_i \int \omega_{err}\,dt + k_p \omega_{err} \qquad (3)$$

Where, M_z is the combined hydrodynamic mass along Z, with ω_{err} being the velocity error in z and is defined as follows:

$$\omega_{err} = \omega_{max}\,(1-Z_{observed}/Z_{actual}) \qquad (4)$$

2. Cruising behavior:

$$\tau_x = M_x \dot{u} + C_u u|u| + k_i \int u_{err} dt + k_p u_{err} \quad (5)$$

Where, Mx is the combined hydrodynamic mass along X, with u_{err} being the velocity error in x and is defined as follows:

$$u_{err} = u_{max} (1 - X_{observed}/X_{actual}) \quad (6)$$

$$\tau_\psi = I_z \dot{r} + C_r r|r| + k_i \int \psi_{err} dt + k_p \psi_{err} \quad (7)$$

Where, I_z is the combined hydrodynamic moment of inertia about Z, with ψ_{err} being the yaw error. PID controllers have been used, k_p, k_d and k_i being the scalar error feedback gains.

2.2. Modeling the deliberative sub-system

As mentioned previously, the *State_Handler* is the most important deliberative model, attributed with two major activities viz., (a) set point update, (b) task sequencing. While the first is associated with changing the reference for the controllers (in the reactive layer) in consistence with the operational specifications, the second one is to govern the execution context of the system. In this paper, an attempt has been made to realize the behavior of the system out on a mission by incorporating only the first deliberation of the *State_Handler*. This is achieved by modeling a state machine as shown in figure 3 and 6 representing the vital actions and transitions evolving from sensory feedback and internal conditions.

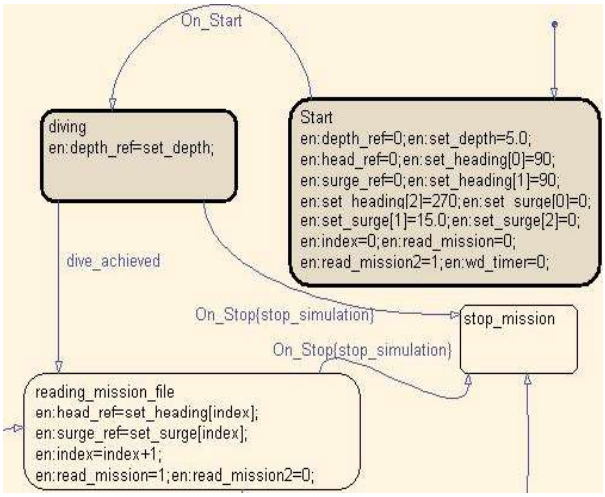


Fig. 5 looking inside the Stateflow control block

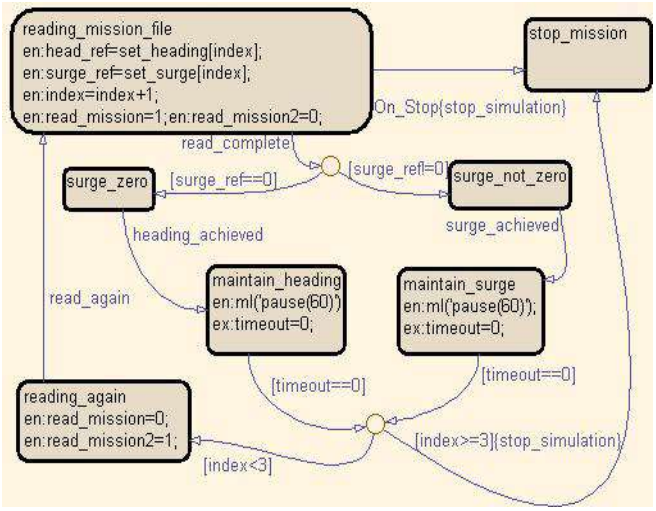


Fig. 6 states responsible for following a trajectory

The hybrid automaton M used for this purpose is defined as follows:

$$M = \{S, \delta, T, A, Op, I, F\} \quad (8)$$

$$\delta : S \times T \rightarrow S \quad (9)$$

$$As : S \rightarrow Op \quad (10)$$

$$At : T \rightarrow Eo \quad (11)$$

$$A = At \cup As \quad (12)$$

Where,

S = {states governing the entire discrete unit}
T = {transitions for a particular state}
δ = state-transition function
Eo = {events generated by the Stateflow machine}
As = {actions being executed while entering /staying into a particular state}
At = {actions executed while a transition takes place}
Op = {output variables}
I = Initial state with default transition
F = Final state of termination

In this regard, it may be mentioned that equations (10) and (11) represent the deliberative activities of the *State_Handler*. The state-machine controller uses three important states for carrying out its decisive functions. While in *start state*, it initializes the set-point variables, local variables and clears flags thereby, making the system ready for the mission. The *diving state* mainly sets the desired depth to be achieved and notifies the *read_mission_file state* when desired depth has been achieved by raising the *dive_achieved event*. It is the *read_mission_file state*, which is responsible for parsing the entire mission file and updating the reference variables for heading and surge controllers after each segment of trajectory is covered. The end of file (i.e. completion of mission) is verified at the junction associated with *maintain_heading* as well as *maintain_surge states* where the condition index>=3, leads to a transition to *stop_mission state*, thereby executing the associated transition-action viz., *stop_simulation*, resulting in complete termination of the system operation.

3. Results and discussion

Simulation results are represented by figures 8 and 9, wherein the simulated mission remains as aligning itself at 90^0 heading, subsequently going for a surge of 15 meters in the same direction and then again align at 270^0 heading. Settling time for each correction has been set to 5 seconds. The first segment (figure 8) shows a heading correction for 90 degrees with zero surge from t=0.25 to t=1.0 secs. During the second segment, heading is maintained at 90 degrees with surge (figure 9) shooting off for 15 meters and staying at the same position from t=1.0 to t=2.0 secs. Consecutively in the third segment, surge controller shoots off for a zero surge at t=2.0 secs while heading changes to 270 degrees from t=2.0 to t=3 secs. This obviously establishes the dynamic coordination between the event-based deliberations of the *State_Handler* and the continuous responsiveness of the reactive controllers.

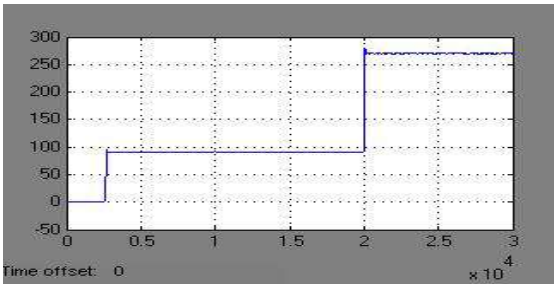


Fig. 8 plot for data obtained from heading controller

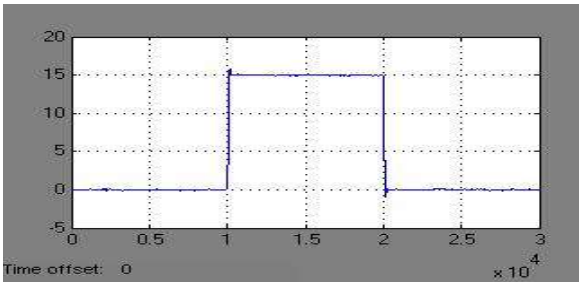


Fig. 9 Plot of data from surge controller

4. Conclusion

A model has been discussed for representing hybrid reactive-deliberative architecture adopted for the control system of an AUV. A significant approach through hybrid modeling methodology has been given to understand the logical organization of the various tasks (or behaviors) of the system, which remains the essential core of the computational architecture. Hybrid modeling technique combines the essential aspects of the computational theory of finite state automaton as well as conventional control theory. It is useful in the sense that it establishes the dynamic switching from one policy to another thereby governing the set of reactive behaviors of the system, in order to achieve a particular goal. Moreover, the overall dynamic behavior of the system has been further established through simulation results as obtained for transient response of reactive controllers governed by the deliberative activities of an event-based finite state machine. Therefore, it is the ultimate methodology to depict the dynamic effect of the hybrid reactive-deliberative control architecture adopted for the system and as discussed in the paper.

Acknowledgements

The authors would like to express their gratitude to the Ministry of Earth Sciences (MoES), Govt. of India, for sponsoring this project. The authors would also like to acknowledge the support of the entire Robotics and Automation group for their cooperative and coordinated efforts towards making the AUV sea-worthy.

References

1. J. Yuh; Design and Control of Autonomous Underwater Robots: A Survey, Autonomous Robots, 8, 7-24, 2000.

2. B. Vinayagasundaram; S. K. Srivastav; Implementation of hybrid software architecture for Artificial Intelligence System, International Journal of Computer Science and Network Security, Vol. 7 No. 1, 2007.

3. G. Dudek; M. Jenkin; Computational principles of mobile robotics, Cambridge University Press, Cambridge, UK, 2000, pp 149-166.

4. G. A. Bekey; Autonomous Robots: from Biological Inspiration to Implementation and Control, The MIT Press Cambridge, Massachusetts London, England, 2005, pp 97-122.

5. A. Oreback; H. I. Christensen; Evaluation of Architectures for Mobile Robotics, Autonomous Robots, 14,33-493, 2003.

6. Y. Kim; J. Lee; S. Park; B. Jeon; P. Lee; Path Tracking Control for Underactuated AUVs Based on Resolved Motion Acceleration Control, Proceedings of the 4th International Conference on Autonomous Robots and Agents, Wellington, New Zealand, Feb 10-12, 2009.

7. Y. Wang; W. Yan; W. Yan; A Leader-Follower Formation Control Strategy for AUVs Based on Line-of-Sight Guidance*, Proceedings ofthe 2009 IEEE International Conference on Mechatronics and Automation, Changchun, China, August 9 – 12, 2009.

8. W. Naeem; R. Sutton; S. M. Ahmad; R. S. Burns; A Review of Guidance Laws Applicable to Unmanned Underwater Vehicles, The Journal of Navigation , U.K. 56, 15–29, 2003.

9. T. Levin; I. Levin; Hybrid Systems Modeling in Learning Science and Technology, Journal of Computers in Mathematics and Science Teaching, 21 (4), 313-330, 2002.

10. M. Youjie; C. Deshu; Z. Xuesong; Hybrid Modeling and Simulation for the Boost Convertor in Photovoltaic System, Second International Conference on Information and Computing Science, 2009.

11. T. I. Fossen; Guidance and Control of Ocean Vehicles, John Wiley and Sons, UK, 1994, pp 48-54.