2005

HIERARCHICAL HYBRID-MODEL BASED DESIGN, VERIFICATION, SIMULATION, AND SYNTHESIS OF MISSION CONTROL FOR AUTONOMOUS UNDERWATER VEHICLES

Siddhartha Bhattacharyya
University of Kentucky, sidhib@yahoo.com

ABSTRACT OF DISSERTATION

Siddhartha Bhattacharyya

The Graduate School
University of Kentucky
2005

HIERARCHICAL HYBRID-MODEL BASED DESIGN, VERIFICATION, SIMULATION, AND SYNTHESIS OF MISSION CONTROL FOR AUTONOMOUS UNDERWATER VEHICLES

ABSTRACT OF DISSERTATION

A dissertation submitted in partial fulfillment
of the requirements for the degree of Doctor of Philosophy in the
College of Engineering
at University of Kentucky

By
Siddhartha Bhattacharyya

Lexington, Kentucky

Co-Director: Dr. Ratnesh Kumar, Professor of Electrical Engineering
and Dr. L.E. Holloway, Professor of Electrical Engineering

Ames, Iowa

Lexington, Kentucky

2005

ABSTRACT OF DISSERTATION

HIERARCHICAL HYBRID-MODEL BASED DESIGN, VERIFICATION, SIMULATION, AND SYNTHESIS OF MISSION CONTROL FOR AUTONOMOUS UNDERWATER VEHICLES

The objective of modeling, verification, and synthesis of hierarchical hybrid mission control for underwater vehicle is to (i) propose a hierarchical architecture for mission control for an autonomous system, (ii) develop extended hybrid state machine models for the mission control, (iii) use these models to verify for logical correctness, (iv) check the feasibility of a simulation software to model the mission executed by an autonomous underwater vehicle (AUV) (v) perform synthesis of high-level mission coordinators for coordinating lower-level mission controllers in accordance with the given mission, and (vi) suggest further design changes for improvement.

The dissertation describes a hierarchical architecture in which mission level controllers based on hybrid systems theory have been, and are being developed using a hybrid systems design tool that allows graphical design, iterative redesign, and code generation for rapid deployment onto the target platform. The goal is to support current and future autonomous underwater vehicle (AUV) programs to meet evolving requirements and capabilities. While the tool facilitates rapid redesign and deployment, it is crucial to include safety and performance verification into each step of the (re)design process. To this end, the modeling of the hierarchical hybrid mission controller is formalized to facilitate the use of available tools and newly developed methods for formal verification of safety and performance specifications. A hierarchical hybrid architecture for mission control of autonomous systems with application to AUVs is proposed and a theoretical framework for the models that make up the architecture is outlined..

An underwater vehicle like any other autonomous system is a hybrid system, as the dynamics of the vehicle as well as it's vehicle level control is continuous whereas the mission level control is discrete, making the overall system a hybrid system i.e., one possessing both continuous and discrete states. The hybrid state machine models of the mission controller modules is derived from their implementation done using TEJA, a software for representing hybrid systems with support for auto code generation. The verification of their logical correctness properties has been done using UPPAAL, a software tool for verification of timed automata a special kind of hybrid system. A Teja to Uppaal converter, called *dem2xml*, has been created at Applied Reserarch Lab that converts a hybrid (timed) autonomous system description in Teja to an Uppaal system description. Verification work involved developing abstract models for the lower level

vehicle controllers with which the mission controller modules interact and follow a hierarchical approach: Assuming the correctness of level-zero or vehicle controllers, we establish the correctness of level-one mission controller modules, and then the correctness of level-two modules, etc. The goal of verification is to show that any "valid" meaning for a mission formalized in our research verifies the safe and correct execution of actions. Simulation of the sequence of actions executed for each of the operations give a better view of the combined working of the mission coordinators and the low level controllers. So we next looked into the feasibility of simulating the operations executed during a mission. A Perl program has been developed to convert the UPPAAL files in .xml format to OpenGL graphic files. The graphic files simulate the steps involved in the execution of a sequence of operations executed by an AUV. The highest level coordinators send mission orders to be executed by the lower level controllers. So a more generalized design of the highest level controllers would help to incorporate the execution of a variety of missions for a vast field of applications. Initially, we consider manually synthesized mission coordinator modules. Later we design automated synthesis of coordinators. This method synthesizes mission coordinators which coordinate the lower level controllers for the execution of the missions ordered and can be used for any autonomous system.

KEYWORDS: Autonomous Systems, Hybrid Systems, Verification, Hierarchical Architecture, Coordinator Synthesis

Siddhartha Bhattacharyya

09/15/05

HIERARCHICAL HYBRID-MODEL BASED DESIGN, VERIFICATION, SIMULATION, AND SYNTHESIS OF MISSION CONTROL FOR AUTONOMOUS UNDERWATER VEHICLES

By

Siddhartha Bhattacharyya

Dr. Ratnesh Kumar

Director of Dissertation

Dr. YuMing Zhang

Director of Graduate Studies

09/15/2005

RULES FOR THE USE OF DISSERTATIONS

Unpublished dissertations submitted for the Doctor's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the dissertation in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

A library that borrows this dissertation for use by its patrons is expected to secure the signature of each user.

Name	Date

DISSERTATION

Siddhartha Bhattacharyya

The Graduate School

University of Kentucky

2005

HIERARCHICAL HYBRID-MODEL BASED DESIGN, VERIFICATION, SIMULATION, AND SYNTHESIS OF MISSION CONTROL FOR AUTONOMOUS UNDERWATER VEHICLES

DISSERTATION

A dissertation submitted in partial fulfillment
of the requirements for the degree of Doctor of Philosophy in the
College of Engineering
at University of Kentucky

By
Siddhartha Bhattacharyya

Lexington, Kentucky

Co-Director: Dr. Ratnesh Kumar, Professor of Electrical Engineering
and Dr. L.E. Holloway, Professor of Electrical Engineering

Ames, Iowa

Lexington, Kentucky

2005

ACKNOWLEDGMENTS

The following dissertation, while an individual work, benefited from the insights and direction of several people. First, my Dissertation Co-Chairs, Dr. Ratnesh Kumar and Dr.L.E.Holloway, exemplifies the high quality scholarship to which I aspire. In addition, they provided timely and instructive comments and evaluation at every stage of the dissertation process, allowing me to complete this project on schedule. Next, I wish to thank the complete Dissertation Committee, and outside reader, respectively: Dr. Zhang, Dr. Manivannan and Dr. Baxter. Each individual provided insights that guided and challenged my thinking, substantially improving the finished product.

In addition to the technical and instrumental assistance above, I received equally important assistance from family and friend (Ganesh). My wife, Srabosti, provided on-going support throughout the dissertation process, as well as technical assistance critical for completing the project in a timely manner. My father, (Dr. S.K. Bhattacharyya), mother (Purabi Bhattacharyya), sister (Suparna chatterjee) and brother in law (Debasish chatterjee) instilled in me, from an early age, the desire and skills to obtain the Ph.D. Finally, I wish to thank the respondents of my study (who remain anonymous for confidentiality purposes). Their comments and insights created an informative and interesting project with opportunities for future work.

Table of contents

Table of Figures

1 Introduction

In this research, our goal has been to develop hierarchical hybrid mission control architecture for autonomous systems illustrating its application to autonomous underwater vehicle (AUV), verify the logical correctness of the controller designed, look into the feasibility of simulating the operations executed by the AUV, and automate controller synthesis. The correct operation of a system we design is a requirement. The challenge to develop a hierarchical hybrid mission controller for underwater vehicle which facilitates modeling, verification, simulation and automated synthesis of coordinators has lead to research in this area. We have worked and are working on these issues with Applied Research Laboratory (ARL) at Pennsylvania State University (PSU) who have designed autonomous underwater vehicles for over 50 years primarily under the support of the U.S. Navy through the Office of Naval Research (ONR).

The control tasks for an underwater vehicle or for an autonomous system can be divided into lower level control, concerned with continuous dynamics and a higher-level mission coordinator/coordinators, which is discrete, either event-driven, or time-driven. The mission coordinators contain both sequence coordinator and timed coordinator for sequential execution and timed execution of various operations of the mission. Thus the overall system is a hybrid system containing both continuous and discrete states. (See chapter 2 for an introduction on hybrid systems.) Design and verification of hybrid systems is highly challenging task, owing to the sophistication and complexity of design and verification. To simplify the complexity of design, researchers at ARL worked with us to formulate a hierarchical control architecture upon which the mission controller design is based. Similar hierarchical architectures built earlier didn't facilitate the development of a model which can be easily put to verification. Our hierarchical hybrid mission control architecture not only facilitates the design of a complex mission controller, it also facilitates verification (done hierarchically in a bottom-up fashion chapter 3), simulation and also the automated synthesis of the highest level mission coordinators.

Our method can be used for other autonomous systems. The basic idea is to hierarchically decompose missions into sequence of operations, and operations into sequence of

behaviors, and behaviors into sequence of vehicle maneuvers. Then we need to design a behavior-controller for each behavior that does appropriate coordination of appropriate vehicle-manuever controllers, an operation-controller for each operation that does appropriate coordination of appropriate behavior controllers, and a coordinator for each mission specification (untimed, timed, and safety) that does appropriate coordination of appropriate operations controllers. We have illustrated our approach through a specific example of mission, namely, a search mission. The same philosophy can be utilized in designing mission controller for other types of missions, such as surveillence and attack. So although the behavior/operation/coordinator controllers that will be designed will vary from mission type to mission type, the approach of the whole of the mission-controller remains the same for all autonomous vehicles for all missions. The generalized approach to automate the synthesis of mission coordinators can be used to any kind of application for any kind of AUV.

Control of autonomous underwater vehicles (AUVs) present specific issues related to automatic control but also classic concerns of real time and high level programming. Several approaches to the design of controllers for AUVs have missed the classical concerns which are a necessity because of the environment in which the AUV needs to operate. Control systems for AUVs have several communicating subsystems/modules. These subsystems/modules need to interact among themselves to successfully execute a mission within satisfactory real time bounds. They constantly react with the environment so they must react in real time to sensory information, thus the need for considering classical issues. Our hierarchical hybrid mission control architecture takes into consideration the real time and high level programming concerns as well. Each of the subsystems developed is a hybrid system which takes care of real time issues by including continuous variables which implement real time or time bounded constraints. The subsystems have been implemented using a high level programming environment provided by Teja.

Verification of a mission control architecture developed for an AUV or for any autonomous system has been neglected because of the lack of such a simplified model. Our hierarchical hybrid mission control architecture supports verification with ease because of the simplified hybrid model on which it is based. For formal verification we

use a graphic tool called Uppaal as opposed to Esterel. This is because Esterel does formal verification of control laws with a complicated method of verification and requires very careful coding in Esterel whereas Uppaal, a graphical tool, models a hybrid automaton easily and gives diagnostic traces efficiently. The verification method used abstracts the subsystem models. This method of using abstractions might miss upon some situations which can be caught in an environment in which all the coordinators and the controllers work together. Such a combined approach can be developed if we can simulate the working of the hierarchical hybrid mission control architecture as a whole. Thus the simulation built simulated the combined working of the subsystems to execute an operation which involves the execution of a sequence of actions.

Mission coordinators at the highest level of the hierarchical hybrid mission control architecture pass control to the lower level controllers for the execution of mission orders. A general design of the mission coordinators will help in using the same mission coordinators for different kind of applications for AUVs other than that illustrated in this dissertation. So we finally automated the design of the mission coordinators.

The objective of this PhD was to develop a control architecture for an autonomous system which is a hybrid system, illustrating it with an autonomous underwater vehicle. Then, we develop a method to verify the logical correctness of the controller designed for the AUV. Then simulate the execution of missions by the AUV. Finally, to automate coordinator synthesis for AUV and enhance the architecture.

In our approach the initial mission controller modules have been developed using TEJA NP networking software platform [22] by the ARL researchers. TEJA supports the design of interacting hybrid controller modules, and offers the autocode generation capability. For verification purposes, these modules can be converted to modules of the UPPAAL [23] verifier. UPPAAL is hybrid system verification software which can be used to verify the safety and correctness of mission controller. In order to proceed with the verification, the first task is to develop an extended hybrid state machine based model of the mission controller, which we have accomplished (see chapter 2). Then we formalized the notion of correctness that demonstrated that any mission can be correctly executed by the mission controller. Also, abstract models of the lower level vehicle controllers (and possibly the underwater sea vehicle) were developed. This we did in consultation with

ARL researchers. Next, a bottom-up approach to verification of logical correctness was formulated and implemented. Verification can be done for something that is already designed, so we use our hierarchical mission controller architecture for search as the model to verify the correctness of the existing design. We simulated the operations executed by the hierarchical hybrid mission control architecture, and we finally accomplished the task of automating the synthesis of a general mission coordinator. As an example the coordinator synthesis works correctly to synthesize the current coordinators built at ARL, by the experience gained we suggest further modifications in the controller design approach within future work. In chapter 2 we discuss hierarchical hybrid mission control architecture and the hybrid system model, in chapter 3 we discuss the approach used for verification, then in chapter 4 we discuss the simulation of the missions and in chapter 5 we discuss the automated synthesis of coordinators. In chapter 6 we conclude with future work.

2 Hierarchical Hybrid Model-based Architecture

Our goal is to develop a mission control architecture for autonomous underwater vehicles (AUVs) that facilitates the modeling and subsequently, verification of the logical correctness of the mission controller/AUV. The mission controller has been under development at the Applied Research Laboratory at the Pennsylvania State University. The design of the architecture in which the mission controller is being developed has benefited from the discussions with the collaborators from Iowa State University and University of Kentucky.

Many of the dynamical systems that need to be controlled, called *plants*, are complex, large-scale, highly nonlinear, time-varying, stochastic, and operate in an uncertain and unpredictable environment. As a result of these characteristics, these dynamical systems are not amenable to accurate modeling. Hence, conventional control techniques that are *model-based*, i.e., rely on the plant model, are not suitable for the controller design for such systems. Also, the control requirements for complex systems are far beyond those for conventional control and include additional requirements such as reconfigurability, learning capability, safety, failure, and exception handling, capability to manage dynamically changing mission goals, multi-system coordination, and increased autonomy. The inadequacy of conventional control techniques for the reasons described above has led to research into the "nonconventional" control techniques, also called *intelligent control*. Intelligent control offers an alternative to conventional control for designing controllers whose structure and consequent outputs in response to external commands and environmental conditions are determined by empirical evidence, i.e., observed input/output behavior of the plant, rather than by reference to a mathematical or model-based description of the plant. For an exposure to intelligent control techniques readers are referred to the edited volumes [32][33][34][35]. There is little to be gained by intelligent control when the plant model is well known and control requirements fall within the scope of conventional control. For this reason, the control is generally hierarchically structured, where at the lower level, conventional control is exercised, whereas at the higher level intelligent control is used, which is usually inherently nonlinear. Several techniques for such nonlinear controller design have been proposed in

literature, which include expert systems, fuzzy logic systems, artificial neural networks, genetic algorithms, a survey of which has been done in [84].

Although various alternative techniques for intelligent control are being actively researched, there is little research effort directed toward the design of intelligent control architectures. One such architecture by Saridis [36] is hierarchical with three layers: the execution layer at the bottom, the coordination layer in the middle, and the organization layer at the top. Meystel [37]has proposed a nested *hierarchical control architecture* for the design of intelligent controllers. A *model-based autonomous systems architecture* by Zeigler–Chi [38] consists of models of planning, operations, perceptions, diagnostics, and recovery. An architecture consisting of a network of intelligent nodes is proposed by Levis [44] as a model for distributed intelligent system. Intelligence in each node is the consequence of its five-stage model, namely, 1) situation assessment, 2) information fusion, 3) task processing, 4) command interpretation, and 5) response selection. Another architecture, called real-time control system (RCS) reference model architecture is by Albus [4]. RCS is also arranged in a hierarchy, where each node in the hierarchy performs sensor processing, value judgment, world modeling, and behavior generation at a level of abstraction and resolution appropriate for the position of the node in the intelligent control hierarchy. Other control computation architecture, called cerebellar model articulation controller (CMAC), was also proposed by Albus [40], [41] to model control computations in intelligent biological systems. A *structure-based hierarchical architecture* is proposed by Acar–Ozguner [39]. It embeds intelligence in control via a special hierarchical organization based on the physical structure of the system. Another architecture, called *subsumption architecture* (Figure 1), is by Brooks [26]. This architecture is based on the idea of levels of increasing competence of an intelligent system, which need to be identified in the beginning of the design phase. *Subsumption* is a method of transforming a robot's control architecture into a set of task-achievement behaviors or competences represented as separate layers. Individual layers work on individual goals concurrently and asynchronously. Layers are organized hierarchically allowing higher layers to inhibit inputs or suppress outputs of lower layers. This constitutes the coordination method. The architecture can be built incrementally adding layers in different phases. Each layer is composed of one or more Augmented Finite State

Machines (AFSM), and depending on sensory information the layer can be active or not. When a layer is active, its output suppresses all outputs from the layers below taking the control of the vehicle. The layer can remain active for a period after the activation conditions finishes.

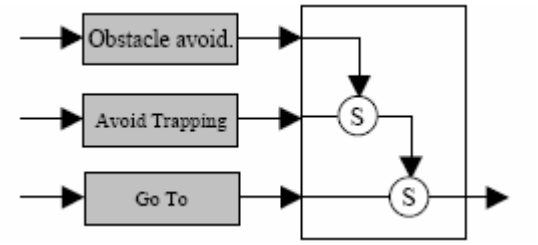


Figure 1: Subsumption architecture

A *Schema-based approach* by Arkin [46], and Warren [47] contains a *Motor schema* as the basic unit from which complex actions are constructed. Each schema operates as a concurrent, asynchronous process initiating a behavior. Motor schemas react proportionally to sensory information perceived from the environment. In order to give more priority to some schemas the output vector is multiplied by a gain value. Critical behaviors are prioritized by assigning them higher gain values. The coordination method consists of vector summation of all motor schema outputs and normalization shown in Figure 2.

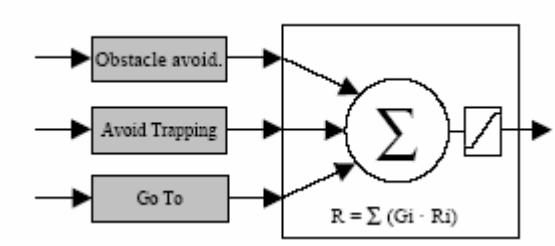


Figure 2: Schema based architecture

Another architecture called the *Process Description Language (*PDL) (Figure 3) proposed by Steels [48] is a cooperative dynamics architecture where many active processes operate in parallel. Processes represent behaviors taking information from sensors to generate a control action if needed. No control mechanism gives precedence over processes. Instead, each process has a certain influence on some variables, typically the motor speeds. At each time, fixed quantities, of each behavior, are added or subtracted to

the previous output value. All behaviors are always operational. The merged output taken depends on which behavior process influences more over the others. Emergent behaviors appear by the effect of this merging. PDL works by manipulating derivatives of the variables; this implies that a very fast control loop must be used to prevent the system from becoming unstable.

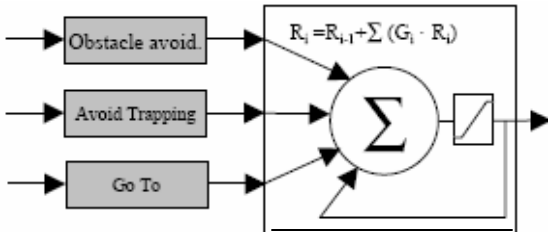


Figure 3: Process description language architecture

Action-selection dynamics, (Figure 4) by Maes [49] uses a dynamic mechanism for behavior selection. Coordination is achieved by competition. There is a set of competence modules that represent behaviors and behaviors are executed when conditions within a condition list is satisfied. Modules have a level of energy that is modified by different sources. Firstly, the sensor signals provide energy to the modules depending on the environment perceived. If the module has in its add list one of the goals of the architecture, more energy is transferred. Finally, the energy of the modules is spread positively along predecessor modules and successor modules, and negatively along conflictors. The module that will be executed is the one that accomplishes the condition list, has the maximum activation energy and this is above a threshold.

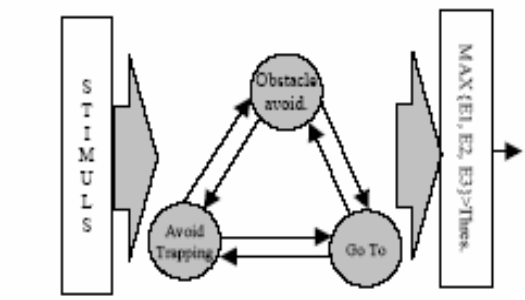


Figure 4: Action selection dynamics architecture

Ridao [59] uses a three layer hybrid architecture for a mobile robot. The deliberative layer at the top based on planning (reasoning and prediction take place here), then the control execution layer (switches behaviors on/off), and the function reactive layer (uses fuzzy logic for behavior description). The paper by Carreras [50] proposes a Hybrid Coordination method for Behavior-based Control Architectures.

Many of the architectures used the high capability of Reinforcement Learning (RL) [51] for robot learning to implement behaviors using this technique. RL has been applied to various Behavior-based systems, most of them using Q_learning [52]. In some cases, the RL algorithm was used to adapt the coordination system [53] [54]. On the other hand, some researchers have used RL to learn the internal structure of a behavior, mapping the perceived states to robot actions [55], [56], [57]. The work presented by Mahadevan [55] demonstrated that the decomposition of the whole agent learning policy in a set of behaviors, as Behavior-based robotics proposes, simplified and increased the learning speed. The approach taken in [50] is a continuous implementation of the Q_learning algorithm. The behaviors were learnt online by means of Reinforcement Learning. Generalization between states and actions was achieved by a feed-forward neural network which approximates the Q_function. Direct Q_learning [50] (backpropagation) was used to train the network. A *decoupled, distributed AUV control architecture* (Figure 5) explained in [58] consists of high level process and behaviors that run the vehicle. The sensor data is pre-processed to produce virtual sensor information available to the behaviors. The behaviors receive the virtual sensor information and send votes to the arbiters who send control signals to the low level controllers. The diagram below shows the control architecture

By decoupling the control problem, individual controller design is greatly simplified. In the case of the Oberon vehicle, vertical motion is controlled independently of its lateral motion using two separate PID controllers. These controllers are then tuned to provide the required performance in each case.

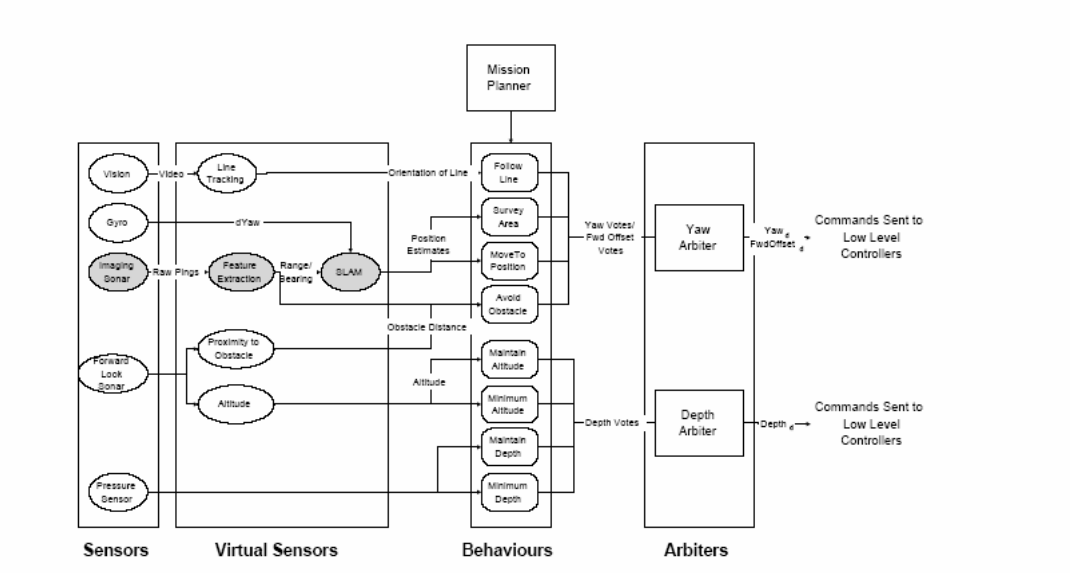


Figure 5: Decoupled distributed AUV control architecture

A *behavior based intelligent control architecture* (Figure 6), for intelligent controllers is by Kumar-Stover [45]. Behaviors determine the manner in which the system reacts to changing external/environmental conditions and thereby executes subtasks of the given mission tasks. The intelligent control architecture is a cascade of subsystems: 1) the perceptor, and 2) the responder. The perceptor extracts the relevant symbolic information from the incoming continuous sensor signals, while the responder is a discrete event system [45]that computes discrete control actions in response to the discrete inputs from the preceptor.

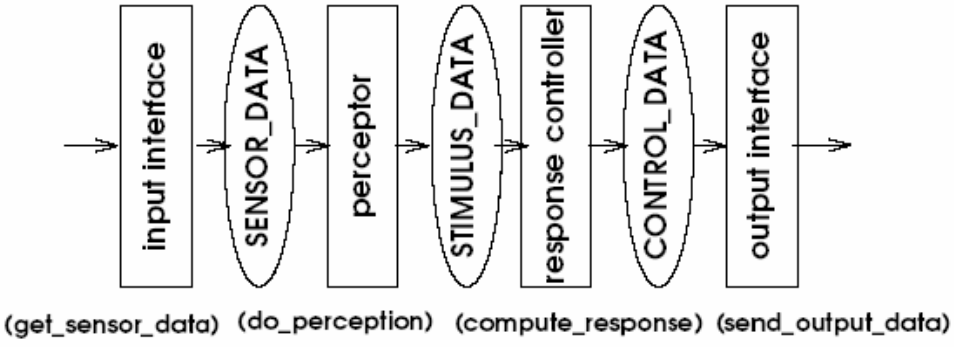


Figure 6: Behavior based intelligent control architecture

Control of autonomous underwater vehicles presents specific issues related to automatic control but also classic concerns of real time and high level programming.

Vehicle control systems for AUVs have several communicating subsystems/modules which need to interact amongst themselves and the environment via sensors to successfully execute a mission within satisfactory real time bounds. Several programming and control architecture have been developed for control of AUVs. Traditionally, artificial intelligence methods are encountered in literature to deal with high level programming; real time and automatic control issues are however, not of first concern in those approaches resulting in unsatisfactory real time performance. The hierarchical approach given in [25] is very rigid and promotes supremacy of a higher level controller restricting low level communications. The Layered reactive approach introduced in [27] lacks a top level supervisor and is organized as a set of communicating software modules. A fruitful adaptation of this approach is State Configured Layer Control [26]. The method used in [28] is to model Robot actions using a robot task concept merging a control law and a logical reactive behavior. At a mission management level, these elementary actions are scheduled using the synchronous programming language Esterel. The method of handling missions however, is not very structured and also while Esterel allows formal verification, the methodology is complicated.

2.1 Proposed Hierarchical Hybrid Mission Control Architecture

While the focus of intelligent control architectures has been the use of "intelligent" technologies such as adaptation, learning, etc., to facilitate safe execution of missions in complex environments, our focus is additionally on real-time operations, automatic code-generation, and semi-automatic verification of safety and progress. To this end, the architecture proposed here is model-based and hierarchical. The models we use are general hybrid dynamical systems, where the enabling conditions and output actions associated with state transitions can be general functions, and real-time constraints are explicitly accounted for.

In our approach the control tasks for an AUV can broadly be divided into lower level control, concerned with continuous dynamics and high-level control, which is typically discrete, and event/time-driven. In this paper, we refer to the lower level of control as the Vehicle Control (VC) and to the higher level of control as Mission Control (MC). The overall system is therefore, a hybrid system containing both continuous and discrete

states. In an attempt to manage the complexity of design, we formulate a hierarchical control architecture upon which the mission controller design is based. This architecture not only facilitates the design of a complex mission controller, it also facilitates the verification and potentially, the automated synthesis of the highest level mission coordinator(s).

The basic idea is to hierarchically decompose AUV missions into sequences of *operations*, operations into sequences of *behaviors*, and behaviors into sequences of vehicle *maneuvers*. A mission/operation can also contain commands for vehicle maneuvers. Then, each level of the hierarchy coordinates the level below it to accomplish specific tasks. The MC design is then accomplished in a bottom up fashion, starting with the design of behavior controllers which coordinate vehicle controllers, moving up to operation controllers which coordinate the behavior controllers; and finally, a coordinator for each type of mission specification (e.g., safety and progress---untimed/timed) which coordinates the operation controllers.

The mission controller modules are developed using TEJA NP networking software tool [22]. TEJA supports the design of interacting hybrid state machines and includes automatic real-time code generation which allows for rapid deployment on the target platform. For verification purposes, the Teja modules specifications are first represented formally and then transformed into a format readable by UPPAAL [23], a hybrid system modeling, simulation, and verification tool. Abstract models of the lower level vehicle controllers (and possibly a underwater vehicle) are developed and also represented in TEJA and UPPAAL. Section 2.2 outlines the hybrid systems model framework that is used to formalize the mission controller modules, section 2.3 describes our hybrid mission control architecture, and section 2.4 describes the tool and procedure for formal verification of safety and performance of a specific mission control system.

Our mission control architecture is designed with semi-automatic (safety and progress) verification in mind. All the levels and modules that make up the hierarchy conform to the interacting controlled hybrid systems model described in Section 2.2; and the tool used to implement the hierarchy allows the conversion of the representation of the hybrid automata into a format that is readable by available verification tools such as HyTech [31] and Uppaal [32]. Following a hybrid system description, Teja facilitates

communication between hybrid subsystems via shared data and event synchronization. Each Teja system must contain a user-defined event dependency table that specifies which subsystems may receive events that are sent from another subsystem. When a Teja subsystem initiates an event, it is passed to all subsystems listed within the event dependency table, causing synchronization. The logic within individual automatons is restricted to use clocks as the only continuous variables and all the continuous dynamics are encapsulated in functions allowing the verification problem to be decomposed into *safety verification* – the verification of the logic of the mission controller; and *progress verification* which is further decomposed into two steps – the verification of the steps leading to the successful completion of each module's goal, and algorithmic verification. (In this dissertation we focus only on the verification of logical correctness.) While existing mission control architectures of AUVs have been deployed successfully, none of them were designed to be used with modern verification tools and techniques without considerable overhead.

The hybrid mission controller is organized hierarchically as shown in Figure 7 below. Modules within a level may communicate with each other and each level in the hierarchy is restricted to command the level immediately below it and send responses to the level immediately above it. All levels in the mission controller hierarchy may assign vehicle commands directly by placing appropriate vehicle commands in the shared database. At the lowest level of the hierarchy is the underwater vehicle (plant) along with the vehicle controllers (VCs). The vehicle and the vehicle controllers have a hybrid state-space (which might, in some vehicles, be a purely continuous state space), and serve as the plant for the higher level mission controller (MC), which is also hybrid in nature.

The vehicle controller and the mission controller communicate through an interface layer symbolically represented by MC2VC (mission controller to vehicle controller) and VC2MC (vehicle controller to mission controller). The MC2VC block also includes a Command Conflict Manager which is responsible for selecting a specific vehicle level command (when more than one exists) according to a static or dynamic priority list or using other methods (such as optimization). This module is included since all modules in the mission controller hierarchy are allowed to assign vehicle commands directly, and so there is a distinct possibility that multiple vehicle commands can coexist.

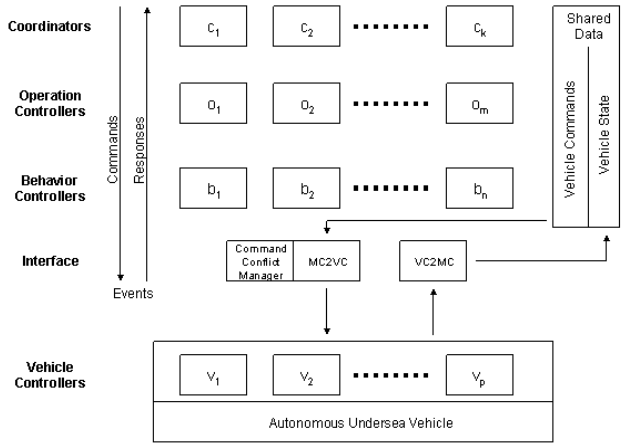


Figure 7: Hybrid Mission Control Architecture

As shown in Figure 8 the mission controller is organized in a three-tier hierarchy, where commands/directives flow down the hierarchy and responses/reports flow the other way. For each command the concerned controller receives a response which makes sure that the subsystems at a lower level are at a ready state to receive the next event and synchronize with the event. The lowest level of the mission controller is comprised of *Behavior Controllers*, where a *behavior* may be thought of as a skill or ability that an autonomous system possesses which enables it to perform specific mission tasks (*thrive*) while remaining safe (*survive*). Behaviors directly interface with the vehicle controllers and are therefore vehicle-centric. They require executions of sequences of vehicle maneuvers. The middle level of the mission control hierarchy consists of *Operation Controllers*, where an *operation* represents a mission segment or phase that is integral to the completion of the overall AUV mission, and are user/mission-centric. They are directly commanded by the user via mission orders and, in turn, command/sequence the behavior controllers to achieve their objectives. The highest level of the mission controller consists of the *Mission Coordinators* which are responsible for sequencing and scheduling operations in order to complete the mission while ensuring the safety of the vehicle. Mission coordinators are typically of two types, safety and progress. Progress coordinator may be separated into untimed and timed.

A mission is defined as a coordinated sequence of operations, each of which is a

sequence of behaviors, and possibly vehicle controller commands. Each behavior is, in turn, a sequence of commands to the vehicle subsystem controllers via the MC2VC interface. Mission termination is defined as aborting a mission due to device, component or vehicle failure. Mission expiration is defined as the timeout on a timed mission as a result of which a timed mission could not be executed. AUV state information is collected by the sensors and transferred by the VC2MC interface periodically to the shared database. This state information is made available to all modules in all levels of the mission controller hierarchy. Similarly, vehicle commands, assigned and manipulated by all levels in the mission controller are stored in the shared database and sent to the AUV by the MC2VC interface.

Command events propagate down the mission controller hierarchy and response events propagate up the mission controller hierarchy via event synchronization. An event is initiated by a particular module and its recipients are controlled by an event dependency table which may be static or dynamic. An event may also initialize parameters within modules in the hierarchy. Command events take the general form $do_{m}^{n}(command, params)$ where *m* is the requesting controller module, *n* is the receiving controller module, *command* is the task to be performed and may take on values such as *initialize, abort*, etc., and *params* are parameters and initial states for the receiving module. Similarly, response events are in the general form $done_{n}^{m}(response, results)$, where *response* is an indication of the completion of the commanded task and may take on values such as *normal*, *abnormal*, etc., and *results* are parameters returned to the requesting module on task completion. Referring again to Figure 7, let *B* denote the set of behaviors, *O* denote the set of operations, and *V* denote the set of vehicle subsystem controllers. A mission, *m* is defined as $m \in M \subset (O+V)^*$, where $(O+V)^*$ is the set of all sequences containing elements of *O* and *V*, and *M* is the set of all possible missions. Similarly, each operation $o_j \in (B+V)^*$, and each behavior $b_k \in V^*$.

2.2 Hybrid Mission Controller for a Survey AUV

Figure 8 shows the details of a specific application of the general AUV mission control architecture to a generic *survey AUV*. The primary mission of a survey AUV is to transit to a user specified location and conduct a survey following a specific pattern in 3D, at a specified speed and depth/altitude. In this example, there are three vehicle controllers (VCs), the *Autopilot* which accepts commands to control the attitude, speed and depth of the AUV; the *Variable Buoyancy System (VBS) Controller* which accepts commands to control the trim and buoyancy of the AUV; and the *Device Controller* which accepts commands to control the various sensors and other devices on board the AUV. Correspondingly, the vehicle state is comprised of the position of the AUV in three dimensions along with the velocity vector, the state of the buoyancy system, and the states of the various sensors and other devices on board.

The lowest level of this mission controller is comprised of four behavior controllers: *Steering* which is responsible for steering the vehicle to a specified location in space and interacts with the Autopilot; *Loiter* which controls the vehicle to loiter at a specific location in space for a specified duration and interacts with the Autopilot and VBS Controller; *Surface/Dive* which commands the vehicle to go to or come off of the surface and interacts with the Autopilot and the VBS Controller; and *Pause* which is used under certain situations to let the vehicle remain at it's current state for a specified duration. These behavior controllers issue appropriate commands for vehicle controllers and monitor their responses, via the vehicle state vector, to achieve their control objectives.

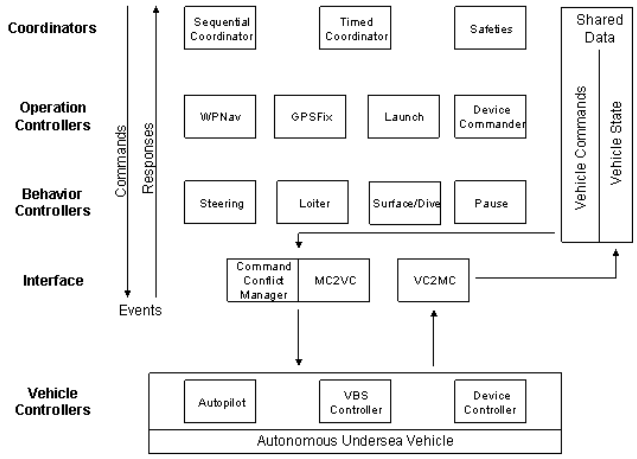


Figure 8: Survey AUV Mission Controller

The behavior controllers are, in turn, commanded by the operation controllers which correspond directly to mission orders that are specified by the user and are described next. The *Launch* operation controller is responsible for bringing the vehicle off of the surface and running at depth with enough forward speed to achieve controllability. This controller interacts with the Autopilot, the VBS Controller, the Device Commander, and the Surface/Dive behavior controller. The *GPSFix* operational controller sequentially commands the AUV to shut off propulsion, rise to the surface, raise the GPS mast, obtain a GPS-aided position fix, retract the GPS mast, and re-launch the AUV. This controller interacts with the Autopilot, the Surface/Dive behavior controller, the Device Commander, the Device Controllers, and the Launch operation controller. The *WaypointNavigator* operation controller controls the AUV to transit to waypoints specified by the mission specification. This controller interacts with Steering, Loiter, and the Device Controller. The *Device Commander* is used to control sensors and devices on the AUV in response to mission orders; this controller interacts with the Device Controllers.

Finally, at the highest level of the AUV mission controller are the mission coordinators of which there are two types: *Progress* and *Safety*, where the progress coordinator is divide into two parts: *Sequential*, and *Timed*. The sequential coordinator is responsible for executing a mission consisting of a sequence of operations; a timed coordinator is responsible for executing a timed sequence of operations; and a safety coordinator ensures safe operation of the vehicle. Timed operations have priority over sequential ones: When a timed operation is due, if necessary, the currently executed sequential operation is suspended until the timed operation has been executed this is taken care of in the model by synchronizing the Timed coordinator with the Sequential coordinator through the *Suspend* signal. Sequential operation is then resumed until the next (if any) timed order is due. Safety coordinator has priority over all other coordinators. When an unsafe condition is detected, the commands from the safety coordinator supercede all other commands and seek to move the vehicle into a safe region or abort the mission if necessary. The architecture has been implemented using the tool Teja.

2.3 Hybrid system model

Hybrid systems are systems which include continuous as well as discrete signals and components. Hybrid systems [24] have been used as mathematical models for many important applications, such as automated highway systems [1], [2], [3], air-traffic management systems [4], [5], [6], embedded automotive controllers [7], [8], manufacturing systems [9], chemical processes [10], robotics [11], [12], real-time communication networks, and real-time circuits [13]. Their wide applicability has inspired a great deal of research from both control theory and theoretical computer science [14 - 21], [10].

Hybrid modeling framework has evolved from the existing modeling ideas of a purely discrete or continuous system in directions such as: Extending the discrete system modeling techniques to develop hybrid automata [72] and hybrid Petri nets [73]; extending the continuous modeling techniques to obtain switched continuous systems [74]; decomposition of a hybrid system into a continuous and discrete subsystem [75]; and the composition of continuous system with a discrete system through compatible interfaces. Some of these modeling concepts are implemented within commercial tools like Simulink/Stateflow. These approaches are developed with the objective to describe dynamical system which exhibit hybrid phenomena and to support their analysis using mathematical methods. The actual modeling approach used depends on the underlying application.

An autonomous system such as AUV, aerial vehicle, is a hybrid dynamical system with both discrete and continuous states. Hybrid systems can be modeled as hybrid automata. A hybrid automaton model captures the evolution of variables over time. The variables will either evolve continuously or in instantaneous jumps. A hybrid automaton is as shown below. This type of modeling formalism will be used to develop underwater vehicle modules.

2.3.1 Controlled hybrid automaton

A *controlled* hybrid automaton is a tuple $\boldsymbol{H} = (Q, \Sigma, U, Y, F, H, I, E, G, R)$ consisting of the following components:

State space: $Q = L \times X$ is the state space of the hybrid automaton, where L is a finite set of locations and $X = \Re^n$ is the continuous state space. Each state Q can be described by $(l, x) \in Q$, where $l \in L$ and $x \in \Re^n$.

Events: Σ is the finite alphabet or event set of ***H***.

Continuous Controls and Parameters: $U = \Re^m$ is the continuous control space consisting of control signals and exogenous continuous-time parameters. $u : [0, \infty) \to U$ denotes a control vector comprised of these parameters.

Outputs: Y is the output space of ***H***, which may consist of both continuous and discrete elements.

Continuous Dynamics: F is a function on $L \times U$ assigning a vector field or differential inclusion to each location and continuous control vector. We use the notation $F(l, u) = f_l(u)$.

Output Functions: H is a set of output functions, one for each location $l \in L$. We use the notation $H(l) = h_l$, where $h_l : X \times U \to Y$ is the output function associated with location $l \in L$.

Invariant conditions: $I \subset 2^{X \times U}$ is a set of invariant conditions on the continuous states, one for each location $l \in L$. We use the notation $I(l) = i_l \subseteq X \times U$. If no i_l is specified for some $l \in L$, then it's default value is taken to be $i_l = X$, indicating the condition will always be satisfied for any continuous state.

Edges: $E \subset L \times \Sigma \times L$ is a set of directed edges. $e = (l, \sigma, l') \in E$ is a directed edge between a source location $l \in L$ and a target location $l' \in L$ with event label $\sigma \in \Sigma$. In addition, $E = E_c \cup E_\Phi$, where E_c and E_Φ represent the controlled and uncontrolled edges, respectively.

Guard conditions: $G \subset 2^{X \times U}$ is the set of guard conditions on the continuous states, one for each edge $e \in E$. We use the notation $G_e = g_e \subseteq X \times U$. If no g_e is explicitly specified for some edge $e \in E$, then it's default value is taken to be $g_e = X$, indicating the guard will be satisfied for any continuous state.

Proaction: Proactions are edges fired when the guard condition is true, i.e. $e = (l, g_e, l')$ no requirement for the occurrence of an event.

Response: Responses are transitions on edges fired as synchronization events from another hybrid automaton expressed as $e = (l, \sigma, l')$.

Reset conditions: R is the set of reset conditions, one for each edge $e \in E$. We use the notation $R(e) = r_e$, where $r_e : X \to 2^X$ is a set-valued map. If no r_e is explicitly specified for some edge $e \in E$, then the default value is taken to be the identity function.

Events can be input or output events. Input events can be local events or events received from other hybrid systems. Local events are events which do not need to synchronize with other hybrid systems for transition to occur.

Definition 1: For $\sigma \in \Sigma$, a σ -step is a binary relation $\overset{\sigma}{\to} \subset Q \times Q$ and we write $(l, x) \overset{\sigma}{\to} (l', x')$ if and only if (a) $e = (l, \sigma, l') \in E$, (b) $x \in g_e \cap i_l$ and (c) $x' \in r_e(x) \cap i_l$. A σ -step need not be taken even if $x \in g_e$, but some σ -step must be taken before it holds that $x \notin i_l$.

Definition 2: Let $\varphi_t^l(x,u)$ be a trajectory of $f_l(u)$ starting from x and evolving for time t. For $t \in \Re^+$, a t-*step* is a binary relation $\overset{t}{\to} \subset Q \times Q$ and we write $(l, x) \overset{t}{\to} (l', x')$ if and only if (a) $l = l'$, (b) $x' = x$ for $t = 0$ and (c) $x' = \varphi_t^l(x,u)$ for $t > 0$ where for $T \in [0, t]$, $\dot{\varphi}_T^l(x,u) \in f_l(\varphi_t^l(x,u))$ and (d) for all $T \in [0, t]$, $x(t) \in i_l$.

Definition 3: A *trajectory* π of ***H*** is a finite or infinite sequence $\pi : q_0 \overset{\theta_0}{\to} q_1 \overset{\theta_1}{\to} \ldots q_{i-1} \overset{\theta_{i-1}}{\to} q_i \ldots$ where $q_i \in Q$ and $\theta_i \in \Sigma \cup \Re^+$. A trajectory is *accepted* by ***H*** if each $q_i \overset{\theta_i}{\to} q_{i+1}$ is a t -step or σ -step of ***H*** , and we denote the space of all such trajectories by ***Tr*** . A *step* of a trajectory refers to a t-step followed by a σ -step.

Associated with the k^{th} step of a trajectory is (a) the time interval of the step, $I^0 = [0, t^1]$ or $I^k = [t^k, t^{k+1}]$ for $k \geq 1$, (b) its duration, $\tau^k = t^{k+1} - t^k$, (c) the associated edge, $e^k = (l^k, \sigma^k, l^{k+1})$, and (d) the state, $q^k = (l^k, x^k(t))$, where l^k is fixed over I^k, $t^k \leq t < t^{k+1}$ and $x^k(t)$ satisfies $\dot{x}^k(t) = F_l^k(x^k(t), u(t))$. Thus, the step can be represented as $(l^k, x^k(t^k+)) \overset{\tau^k}{\to} (l^k, x^k(t^{k+1}-)) \overset{\sigma^k}{\to} (l^{k+1}, x^{k+1}(t^{k+1}+))$ satisfying $x^k(t^{k+1}) \in g_{e^k}$,

$x^k(t^{k+1}-) \in i_{l^k}$ and $x^{k+1}(t^{k+1}+) \in r_{e^k}\left(x^k(t^{k+1})\right)$. Note that we do not exclude the possibility that $\tau^k = 0$, in which case there is only a σ-step.

Definition 4: A *run* of a hybrid automaton ***H*** is the projection to the discrete part of a trajectory in ***Tr***; namely, a finite or infinite sequence $l^0, l^1, l^2, \ldots$ of admissible locations.

We also refer to $x(t) = \sum_{k=0}^{\infty} x^k(t) \mathrm{II}_{I^k}(t)$ where $\mathrm{II}_{I^k}(t)$ is the indicator function of the interval I^k, as the *continuous part* of the trajectory. Note that it is not in general true that $x(t^k+) = x^k(t^k+)$. For instance, if $\tau^k = 0$ and $\tau^{k+1} > 0$ then $x(t^k+) = x^{k+1}(t^k+)$ which is not necessarily equal to $x^k(t^k+)$.

2.3.2 Interacting Controlled Hybrid Automata

In order to cope with complexity of real-life applications it is often convenient to model a hybrid system in a modular fashion as a set of interacting hybrid automata, $\{H^j\}$. Each hybrid automaton in the set is a tuple as before, ***H*****= { $\boldsymbol{H^j}$ =** $(Q^j, \sum^j, U^j, Y^j, F^j, H^j, I^j, E^j, G^j, R^j)$**}**

The interaction among various hybrid autonomous modules takes place through event synchronization and sharing of variables in invariant and guard conditions, as follows.

Invariant Conditions: For each $l \in L^j, I^j(l) \subseteq X^j \times \cup_j U^j$.

Guard Conditions: For each $e \in E^j, g^j(e) \subseteq X^j \times \cup_j U^j$.

The other components of the tuple are analogous to those of the single hybrid automaton defined above.

For an event $\sigma \in \Sigma = \bigcup_j \Sigma^j$, let $In(\sigma) = \left\{ j \mid \sigma \in \Sigma^j \right\}$ be the set of indices of the event sets that contain the event σ. Then each σ -step must be taken *synchronously* by each of the hybrid automata H^j such that $j \in In(\sigma)$. In other words, for each $j \in In(\sigma)$, $(l_1^j, x_1^j) \xrightarrow{\sigma} (l_2^j, x_2^j)$ if and only if (a) $e^j = (l_1^j, \sigma, l_2^j) \in E^j$ (b) $x_1^j \in g_{e^j} \cap i_{l^j}$ and (c) $x_2^j \in r^j{}_{e^j}(x_1^j) \cap i^j{}_{l_1^j}$.

2.4 Teja: Tool for Modeling

Teja NP, is an embedded networking and communications software platform, it includes an Application Development Environment (ADE), a Network Processing Operating

System (NPOS) runtime system and an extensible library of embedded network application building blocks. Teja thus helps model modules which can communicate with each other as is required for the modules in our application to interact.

Teja is a tool that facilitates the graphical design of interacting hybrid automata and includes real-time code generation utilities and supports modularity. TEJA supports two types of models, the data model (input events, states, output actions), and the process model for systems (hybrid automaton).

The data model has five sections: the class name, the superclass, the outputs, the inputs and the constructor/destructor. The superclass section consists of features to add a new superclass and a list for selecting an existing superclass. The output section consists of variables, links and functions. It also consists of facility to either add a new kind of output or select an output from an existing list. The same is available for input and constructor/destructor sections. In addition to that editing capability is available for destructor.

The process model has features to generate code, model continuous state, model discrete state, and transitions. The discrete state models the flow of the continuous state and it does computations that occur at that state. The transitions show the initial state, the final states, the event, the guard condition, and the action that it takes. The drawing part has a list of continuous states and a drawing of the component's hybrid state machine.

The TEJA NP graphical application development environment allows designing finite state machines. This method is used with ease to model a hybrid system such as underwater vehicle. This graphical application helps model common functions in a complex environment: generating and passing information between sender and a receiver. This function applies to complex environment of an underwater vehicle with communicating controller modules at different levels. Example: - The sequence controller sends commands to lower order module Launcher which further sends commands to its lower level modules to get the command executed. This application of TEJA helps to generate, send and receive such a signal.

State machine is used with ease to describe algorithms especially in a real time driven system. Although design is done at very high level, existing design can be easily merged into a design. This is accomplished by direct function calls, message passing and similar

means. TEJA NP applications are designed to run over any operating system. TEJA NP systems automatically outputs codes (C, C++, assembly) and links with existing systems. Thus using this tool gives us the flexibility to use some other tool to veriy the time driven operations as the code generated can be used state diagrams in another verification tool.

Teja allows the creation of a *system architecture* where all the modules required for a particular mission controller are instantiated and initialized, and their interactions are specified via an event dependency table which may be dynamically reset. Automatic code generation ensures that the real-time scheduling needs are met to tolerances far exceeding the mission control application.

Teja allows for abstract class definitions and inheritance so that, when appropriate, generic controller classes may be defined and subclasses may be used to refine and customize the generic controllers to specific applications. Utilities are provided to handle useful functionality such as communications and data handling and parsing. Libraries and utilities are provided for a variety of commonly used platforms and operating systems including Window, Linux, and Solaris. All of these features make Teja an ideal tool for rapid prototyping, testing, and deployment of mission controllers on target vehicle platforms.

2.4.1 Basic structure of Mission Controller Modules

Each controller module is a hybrid automaton whose description in TEJA involves the following items:

Superclass describes the class in Teja the module belongs to which can be any of the following: TejaAux, Teja component, TejaMutex, TejaString, TejaEvent, TejaAlert, TejaFifoQueue

Variables are the local variables of a module. These represent the variables directly accessed by a module which might be its input or output.

Links are the pointer to classes of data structures or finite state machine. In the description the object name is followed by the class name (Object Name (Class Name)). Link is used by one module to access the local variables of another module. It is the input or output of a model based on the application.

Functions are the actions performed by a module based on the guard condition and the event satisfied. Different transitions have different functions depending upon the action it performs.

Constructors are used to initialize the variables used in a module.

Destructors are used to clear the variables so as to free memory but are not used in the application we are using it for.

Hybrid automaton in TEJA contains the description of the discrete state followed by the continuous state flow {state, flow}. Edges are defined by events and guard conditions. For proactions the event is a local event. For responses event is a synchronization event.

The syntax in which the **states** are described is as shown here {state name, (flow), invariant}**.** The **transitions** are given by the description: {initial state, final state, event, guard condition, reset, and action (is the execution of a function or activity in response to an event and guard condition that is satisfied.)}.

A Brief description of all the modules implemented in Teja is explained in section 2.6.2 and a detailed description of all the modules is given in Appendix B.

2.5 Hybrid Automaton Model of Mission Controller Modules

2.5.1 Description of Mission Controller Commands/Responses

In hybrid automaton model, transition from one discrete state to the other takes place on the command/response shown on the edge connecting the states. In the following we describe the commands/responses that appear as transition labels in the various hybrid automata models. In the language of TEJA, commands are known as proaction and responses are known as reaction. Some of the common commands that can be executed by any such underwater vehicles are shown below.

Init: It initializes the module to the Start or Idle state. This signal makes the module ready to receive signal and start functioning.

Abort: This signal terminates the operation executed by a module.

GoTo…, Take…, Process…, Set…: It represents the command sent by higher level controller to lower level controllers to perform a task such as to go to a desired location.

…Done: It represents the response sent by lower level controller to higher level controller when a task is executed to completion.

The other commands/responses are application specific. The commands for the under water vehicle for the application of search is given in Appendix A.

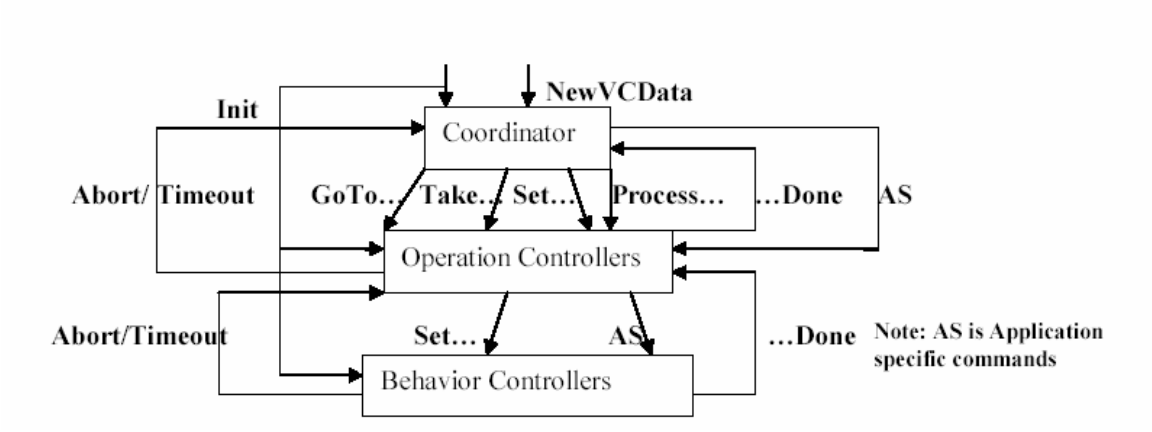


Figure 9: The input/output signals wihtin the controllers

2.5.2 Description of Mission Controller Data Structures

The data structures used for representing states, commands, and responses of a AUV are described here. The following data structures are important for any kind of underwater vehicle. These are general information required by AUV to function correctly and execute all the missions successfully.

VehicleState is a data structure which belongs to the superclass TejaMutex. It contains the status of the vehicle. The required data are: CurrentTime, Shutdown, LaunchDone, and RangeHeartbeat.

BatteryState is a data structure which belongs to the superclass TejaMutex. It contains the status of the battery. The required data are: Volatge, Current, Switches.

NavState is a data structure which belongs to the superclass TejaMutex. It contains data required for navigation. The required coordinates are Latitude, Longitude, Depth, Altitude, Roll, Pitch, Yaw, and Speed.

DeviceState is a data structure which belongs to the superclass TejaMutex. It contains the status of the device. The required data are: MastState, Xvelbtm, Yvelbtm, Zvelbtm,

CTDTemperature, CTDDepth, ADCPBottomLock, VBSFwdPrimed, VBSAftPrimed, VBSFwdMass, VBSAftMass, VBSFwdPressure, VBSAftPressure, and ADCPState.

VehicleCmd is a data structure which belongs to the superclass TejaMutex. It contains commands for a vehicle. The required data are: Behavior and Shutdown.

AutoPilotCommand is a data structure which belongs to the superclass TejaMutex. It contains data required for the autopilot to operate successfully. The datas required are: HeadingMode, HeadingCommand, HeadngRateCmd, DepthMode, DepthCmd, AltitudeCmd, SpeedMode and SpeedCmd.

BatteryCmd is a data structure which belongs to the superclass TejaMutex. It contains commands for a battery. The required data are: Switches.

DeviceCmd is a data structure which belongs to the superclass TejaMutex. It contains commands for a device. The required data are: MastCmd, CTDCmd, SSSCmd, VBSCmd, VBSDepthCmd, VBSFwdCmd, VBSAftCmd, PayloadCmd, and ADCPCmd.

ActionRequest is a data structure which belongs to the superclass TejaMutex. It contains data requests made by modules higher in hierarchy to the ones at lower level. The required data are: GPSRequest, LaunchRequest, WaypointRequest, ControllerRequest, RendezvousRequest, LoiterRequest, PayloadRequest, SteeringRequest, DeviceCommanderRequest and PauseRequest.

TimedOrder is a data structure which belongs to the superclass TejaMutex. It contains data required for timed operations. The required data are: Time, Name, Period and TimingType. OrderPtr is a Link.

ComponentList is a data structure which belongs to the superclass TejaMutex. It contains functions GetComponentName and AddComponent to get the component names wit their ID as assigned internally by Teja. The variables used are: ComponetID, NumComponents and Componentname.

SeqOrd is a data structure which belongs to the superclass TejaMutex. It gives the sequence of the order to be executed. The data required are: Name.

DeviceOrd is a data structure which belongs to the superclass TejaMutex. It gives the orders to be executed by the device. The data required are: Device, Duration and TimimgType. The links are DevCmd and AutCmd.

GPSOrder is a data structure which checks for order to be executed by the GPSFix module. The variables are: CollectSVP and ReturnToStart.

WaitOrder is a data structure which belongs to the superclass TejaMutex. It gives the time of wait for a device or module. The variables are: WaitingTime and TimingType.

LaunchOrder is a data structure which belongs to the superclass TejaMutex. It initiates order. The variables are: ADCPInit and TrimInit.

Files is a data structure which belongs to the superclass TejaMutex. It contains a function CreateLogs to create log files for operations either executed completely or with error. The variables used are errorlogs and execlogs.

Queues is a data structure which belongs to the superclass TejaMutex. It contains the timed and sequenced order. The variables are: TimedOrderQueue and SeqOrderQueue.

2.5.3 Description of the functionalities of mission controller modules

The mission controller modules have been modeled as hybrid automata using Teja NP tool [22]. Transitions between states may be *proactions*, where the transition fires when the guard condition is true, or *responses* which fire on event synchronization from another hybrid automaton. In Teja, the first portion of an event label is either a local label in the case of a proaction, or a synchronization label in the case of a response. The second portion, after the /, represents an output event label that is used to fire enabled response transitions in other modules that are specified in a (static or dynamic) event dependency table for that particular event label. Resets and other initializations may be performed on transitions between states. The hybrid automata modules that make up a particular application therefore interact through event synchronization. Continuous state variables are specified and their flows are defined for each discrete state. An initial state is specified and the Teja tool allows constructors and destructors to initialize and finalize the state variables, and parameters of each automaton. Vehicle state values, on receipt from the interface level, are used to populate Teja data structures which are available to all modules that require access to them. Links to other automata are provided so that public data within them may be accessed and set. These links are used to pass parameters and initial conditions, and retrieve results on event transitions. A brief description of the

mission controller modules follows. A detailed description of the transitions, reset conditions, guards etc. are given in Appendix B

2.5.3.1 Sequential coordinator (SC)

Figure 10 shows the hybrid representation of the Sequential coordinator. The SC module is a sequence coordinator which controls the execution of sequential missions. SC passes control to the operational and behavioral level controllers according to the mission to be executed. The SC consists of the initialization phase, the ready phase and the running phase.

The initialization phase consists of rebooting the system at start up and establishing contact with the vehicle. SC is initially at the ***Idle*** state. It is marked with two concentric circles indicating initial state. When event *Init* occurs at time t>=1 the SC transitions to the state ***WaitForVCComms*** (The inequality condition given by the transition function is the guard condition.). At ***WaitForVCComms*** SC establishes contact with the vehicle.

During the ready phase the SC becomes ready to receive mission orders. On the event *NewVCData* SC transitions to the ***run*** state. The SC check for orders in the queue at the ***run*** state every 1 second.

During the running phase the SC executes the mission ordered by passing control to the lower controllers in the hierarchy. The SC passes control to the DeviceCommander controller by outputting the event *SetDevice* and transitions to ***DeviceOrder*** state. The DeviceCommander performs the ordered mission and sends *DeviceDone* event to the SC on completion. The SC goes to the ***run*** state.

The SC passes control to the Waypointnavigator controller by outputting the event *ProcessWP* and transitions to the ***WayPointNavigator*** state. When the AUV reaches the desired location it sends *WPDone* event to the SC and resets t to 0. The SC goes to the ***run*** state.

The SC passes control to the Rendezvous controller by outputting the event *GoToRendezvous* and transitions to the ***Rendezvous*** state. When the AUV reaches the rendezvous point it sends *RendezvousDone* event to the controller. The SC transitions to the ***run*** state.

The SC passes control to the Payload controller by outputting the event *ProcessPayload* and transitions to the ***Payload*** state. When the AUV reaches the point where it delivers the payload it sends *PayloadDone* event to the SC. The SC then transitions to the ***run*** state.

The SC passes control to the Launch controller by outputting the order *Launch* and transitions to the ***Launch*** state. When the AUV completes executing launch operation like lowering the mast, lowering the AUV below the surface of the water etc. the ***Launch*** controller sends *LaunchDone* event to the SC. The SC transitions to the ***run*** state.

The SC passes control to the GPSFixer controller by outputting the event *GPSFix* and transitions to the ***GPSFixer*** state. When the AUV completes the GPSFix operation by updating the navigation system with the present location it sends the *GPSFixDone* event to the SC. The SC transitions to the ***run*** state.

The SC passes control to the Pause controller by outputting the event *Wait* and transitions to the ***Pause*** state. When the AUV completes the wait operation the Pause controller sends the *WaitDone* event to the SC. The SC then transitions to the ***run*** state.

The SC outputs the event *Suspend* to the controllers which can be suspended and transitions to the ***Suspend*** state. The suspendable controllers in the present structure are: Waypointnavigator, Payload, and Rendezvous. SC remains at the ***Suspend*** state by outputting the event *Abort* and transitions to the ***run*** state on receiving the event *Resume.*

All the states transition to the ***EndMission*** state on the event *Abort*. The ***run*** state can also transitions to the ***EndMission*** state on the event *GoToEndMission*. The SC executes local events *MastUp* and *OnSurface* at ***EndMission*** state to end the execution of missions by bringing the AUV to the surface and raising the mast. This is how the whole of SC module works.

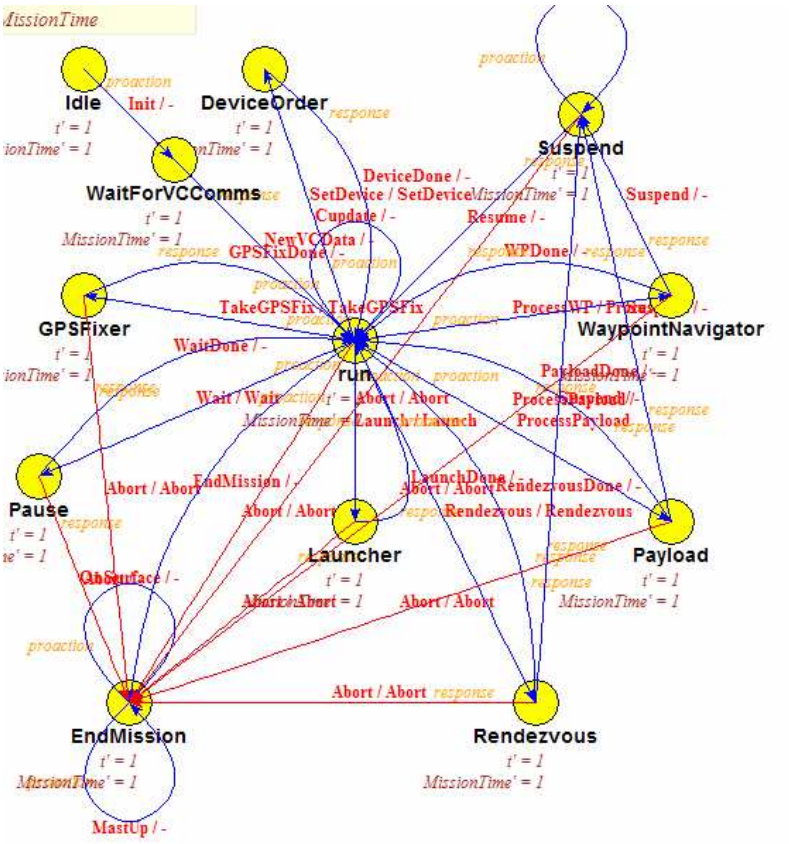


Figure 10: FSM for Sequential coordinator in TEJA

2.5.3.2 Timed Coordinator (TC)

Timed coordinator controls the timed execution of mission operations. It keeps track of the execution time of each operation. It prioritizes the orders according to the time requirement to be met. TC belongs to the topmost level of the mission controller and passes control to the lower level controllers to execute timed sequence of operations.

Initially the TC is in the state ***Start***. On the event *Init* at time t>=1 the TC transitions to the state ***FirstTime.*** On the event *NewVCData* the TC transitions to the ***CheckOrders*** state. The TC keeps checking for the request of a timed order at the ***CheckOrders*** state every 1 second on the local event *NewVCData*. The TC transitions to ***Decide*** state on the event *NewOrder*. The TC transitions to ***Wait4Suspend*** state by outputting the event *Suspend* and keeps checking whether the SC is suspended or not. TC transitions from ***Wait4Suspend*** to ***CheckOrders*** on *Timeout.* Time out occurs when the time for the timed

mission expires before suspending the SC. After suspending the SC the TC transitions to ***Decide*** state from the ***Wait4Suspend*** state on the event *NewOrder*.

The TC passes control to the DeviceCommander controller by outputting the event *SetDevice* and transitions to the state ***Device***. Once the operation on the device is completed the DeviceCommander controller sends *DeviceDone* event to the TC. The TC then transitions to ***Check4Resume***.

The TC passes control to the ***Launcher*** controller by outputting the event *Launch* and transitions to the ***Launch*** state. When the AUV completes executing launch events like lowering mast, bringing the AUV below surface of water etc. the ***Launcher*** controller sends *LaunchDone* event to the TC. The TC then transitions to ***Check4Resume***.

The TC passes control to the ***GPSFix*** controller by outputting the event *GPSFix* and transitions to the ***GPSFix*** state. When the AUV completes the operation to be executed by the GPSFixer it sends the *GPSFixDone* event to the controller. The TC then transitions to ***Check4Resume***.

The TC passes control to the ***Pause*** controller by outputting the event *Wait* and transitions to the ***Wait*** state. When the AUV completes waiting the pause controller sends *WaitDone* event to the TC. The TC then transitions to ***Check4Resume***.

TC checks whether the SC needs to be resumed or not. If TC needs to resume SC, TC transitions to ***CheckOrders*** state by outputting the *Resume* event or else it doesn't output any event.

At each of the states on receiving an *Abort* event the TC outputs an *Abort* event to all the controllers. The TC then transitions to the ***End*** state.

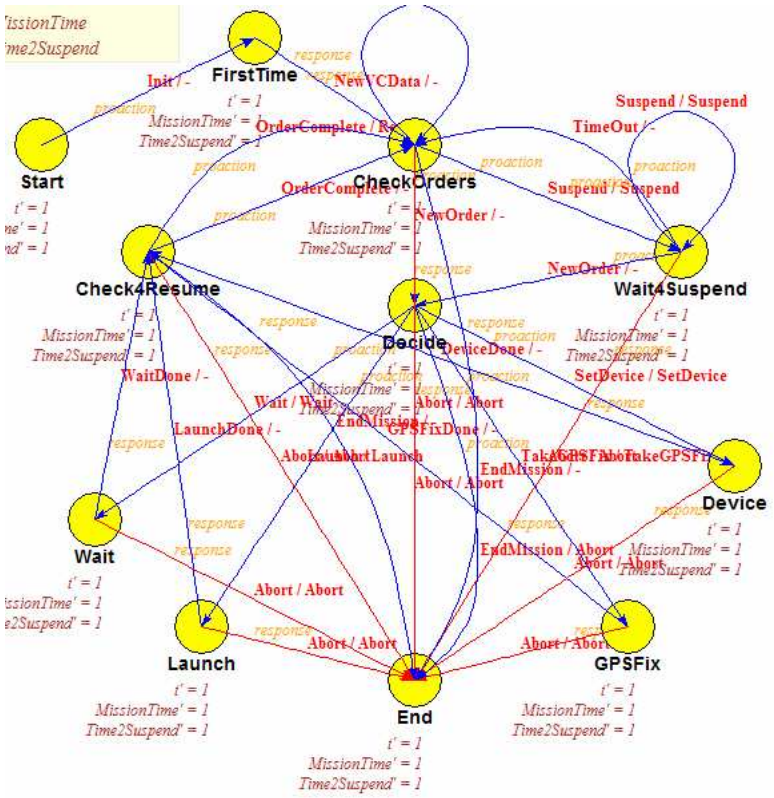


Figure 11: FSM of Timed coordinator in TEJA

2.5.3.3 Safety coordinator

The safeties coordinator is used to check whether the given order can be safely executed or not. The safety coordinator is initially at the ***Start*** state. At t>=1 when the safety coordinator receives the signal *Init* Safety coordinator transitions to the ***Idle*** state. On the event *NewVCData* Safety coordinator transitions to the state ***CheckSafeties*** and executes function to check safe operation. Safety coordinator aborts the mission by outputting *Abort!* event and transitions to the ***SafetyAbort*** state. Safety coordinator tries to correct the altitude if it becomes unsafe by transitioning to ***LowAltitude*** state on the event *AltitudeSafety*. If safety coordinator finds the altitude has been corrected on the event *AltitudeOK* Safety coordinator transitions to the ***CheckSafeties*** state. Safety coordinator transitions to the ***SafetyAbort*** state by outputting the mission *Abort* from the ***LowAltitude*** state if the altitude cannot be corrected to a safe value.

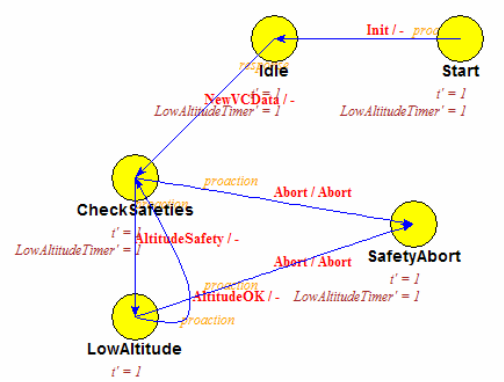


Figure 12: FSM of Safety coordinator in TEJA

2.5.3.4 GPSFixer

Figure 13 shows hybrid representation of GPSFixer operation controller. The GPSFixer controller is used to update the navigation system of the AUV with the present location. The steps involved are to bring the AUV to the surface of the water, raise the vehicle mast and find out the AUVs location. Then to pass control to the Launch controller to lower the mast, lower the AUV from the surface of water and then either return to the original location or to just go to a depth.

The GPSFixer module is initially in the state ***Start***. After initialization GPSFixer controller transitions to the ***Idle*** state on the event *Init.* When the GPSFixer controller receives the event *TakeGPSFix* it starts to execute the sequence of actions involved in the GPSFix mission. GPSFixer transitions to ***GoToSurface*** state on the event *TakeGPSFix.* If the AUV fails to reach the surface in time the GPSFixer controller transitions to ***ReportTO*** state on the event *TimeOut*. On reaching the surface the GPSFixer controller transitions to the ***RaiseMast*** state on the event *OnSurface!*. If the mast is not raised in time on the event *TimeOut* the GPSFixer controller transitions to the ***ReportTO*** state. When the mast is raised the GPSFixer controller transitions to the ***TakeFix*** state on the event *MastUp*. The GPSfixer controller passes control to the Launch controller on the event *Launch* and transitions to the ***ComeOffSurface*** state. The Launch controller lowers the mast and brings the AUV below the surface of water. Then the Launch controller passes control to the GPSFixer controller on the event *LaunchDone.* The GPSFixer transitions to the ***Decide*** state where it decides whether to return back to the original location before starting GPSFix mission or to just go to a particular depth. If the AUV

needs to return to the original location the GPSFixer passes control to the Steer controller on the event *Steer*. Once the AUV reaches the destined location the Steer controller passes control to the GPSFixer controller on the event *SteeringDone*. The GPSFixer transitions to ***Decide*** state. The GPSFixer controller finally ends the mission by transitioning to the Idle state by sending the output GPSFixDone to the concerned controller which can be TC or SC.

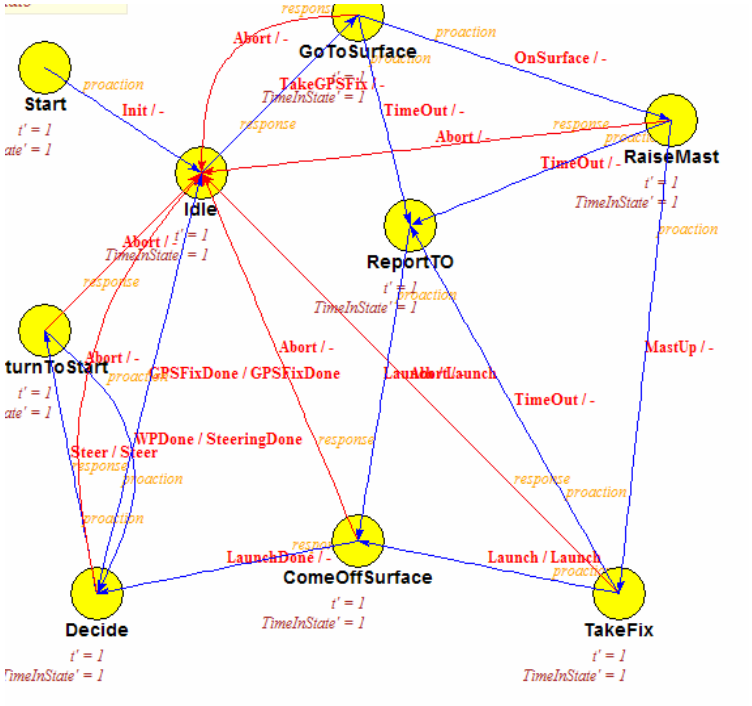


Figure 13: FSM for GPSFixer module in TEJA

2.5.3.5 Launch controller

Figure 14 shows the hybrid automaton representation of the Launch operation controller modeled using the Teja NP tool [22]. The operation executed by the launch operation controller is to lower the mast of the AUV, and lower the AUV from the surface of the water.

Launch operation controller has the initial state as ***Idle***. The Launch controller transitions

to ***RetractMast*** state when it receives the event *Launch* and lowers the mast. If the mast is not lowered within a given time a *TimeOut* event occurs and the Launch controller outputs *Abort* event and transitions to the ***Idle*** state. On the local event *Abort* the launch operation controller also transitions to the ***Idle*** state from ***RetractMast*** state. When the mast is lowered a local event *MastDown* occurs and the Launch operation controller transitions to the ***ComeOffSurface*** state. The AUV is then lowered below the surface of water. If the AUV is lowered successfully the Launch operation controller outputs the event LaunchDone and transitions to ***Idle*** state. If the AUV is not lowered within a given time a *TimeOut* event occurs and the launch operation controller outputs the *Abort* event and transitions to the ***Idle*** state. On the local event *Abort* the launch operation controller also transitions to the ***Idle*** state from the ***ComeOffSurface*** state.

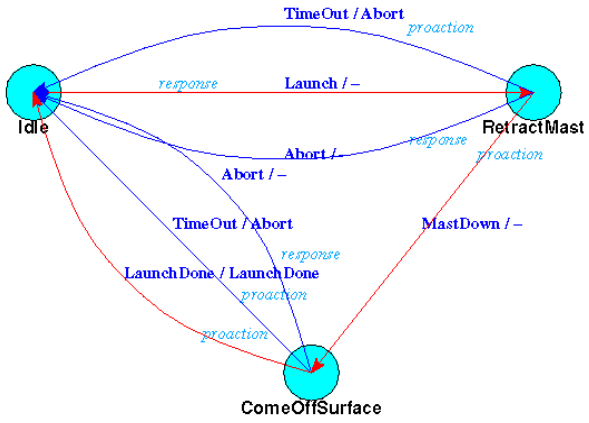


Figure 14: Launch operation controller in TEJA

2.5.3.6 Waypointnavigator

Figure 15 shows the hybrid representation of the Waypointnavigator operation controller. The operation executed by the Waypointnavigator is to navigate the vehicle to a desired point involving timed as well as untimed waypoint navigation. While the AUV is navigating through the water the Waypointnavigator keeps checking the depth of the AUV. If the depth becomes unsafe the waypoitnavigator corrects the depth to a safer value. For a timed waypoint the AUV can loiter if enough time is left between the present location and the desired location.

The initial state of the Waypointnavigator is the ***Start*** state. The Waypointnavigator transitions to the Idle state on the local event *Init.* On receiving the event *ProcessWP* the

Waypointnavigator transitions to the ***Decide*** state. For an untimed waypoint the Waypointnavigator module passes control to the Steer module by outputting the *Steer* event and transitions to ***GoToWaypoint*** state. At this state every second the location of the AUV is checked and if there is any depth trouble the Waypointnavigator transitions to the ***WPDepthProblem*** state. The depth is corrected at this state. If there is failure to correct depth the Waypointnavigator terminates the mission by outputting the *Abort* event. If the depth is corrected the Waypointnavigator transitions to the ***GoToWaypoint*** state on the local event DepthOK. When the AUV reaches the desired location the Waypointnavigator transitions to the ***ReportTO*** state by outputting the *SteeringDone* event. Finally the Waypointnavigator transitions to the Idle state by outputting the *WPDone* event.

For a timed waypoint the Waypointnavigator transitions to the ***TimedWP*** from the ***Decide*** state on the local event *ProcessWP*. The Waypointnavigator transitions to ***GoToLP*** from the ***TimeWP*** state by passing control to the Steer module by outputting the *Steer* event. At the ***GoToLP*** the Waypointnavigator module keeps updating the present location of the vehicle. In case of a depth trouble the depth is corrected by transitioning to the LPDepthProblem state on the local event *DepthProblem*. If the depth problem is not corrected Waypointnavigator transitions to the ***Idle*** state by outputting the *Abort* event. When the depth problem is settled the Waypointnavigator transitions to the ***GoToLP*** state on the local event *DepthOK*. When the AUV has reached the destined location the Steer module passes control to the Waypointnavigator module on the event *SteeringDone* and the Waypointnavigator transitions to the AtWP state. If the AUV reaches the destined location before time the AUV loiters around the desired location. The Waypointnavigator passes control to the Loiter module by outputting the *Loiter* event and transitions to the ***Loiter*** state. When loitering determined by time left to go to destined location is over the Loiter module passes control to the Waypointnavigator by outputting the *LoiterDone* event. The Waypointnavigator transitions to the ***LoiterDone*** state. The Waypointnavigator transitions to the ***LoiterDone*** state from the ***AtWP*** state if enough time is not left for loitering. Then the Waypointnavigator transitions to the ReportTO state on the local event WPDone. Finally the Waypointnavigator transitions to the Idle state by outputting the *WPDone* event.

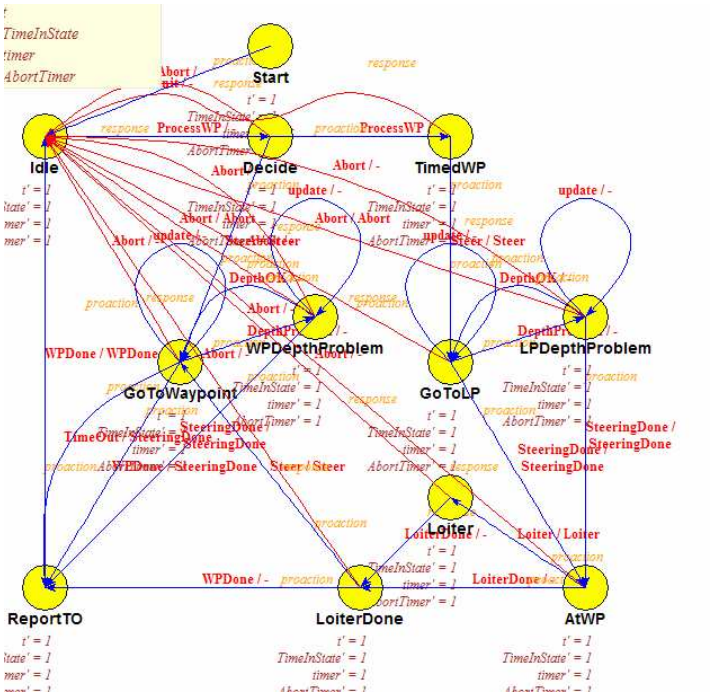


Figure 15: FSM of Waypointnavigator in TEJA

2.5.3.7 Rendezvous

Figure 16 shows the hybrid representation of the Rendezvous module. The rendezvous module is used to guide the vehicle to go to rendezvous point.

The initial state is the ***Start*** state. On the local event *Init* the Rendezvous module transitions to the ***Idle*** state. On receiving the *Rendezvous* event the Rendezvous module transitions to the ***Decide*** state. If the rendezvous point is not specified the Rendezvous module passes control to the Loiter module by outputting the *Loiter* event. After loitering for specified time the Rendezvous module transitions to the ***LoiterDone*** state on the event *LoiterDone*. The Rendezvous module finally goes to the ***Idle*** state by outputting the event *RendezvousDone*.

If the rendezvous point is specified the Rendezvous module transitions to the ***GoToRendezvous*** state by passing control to the Waypointnavigator by outputting the event *ProcessWP*. Once the rendezvous point is reached the control is passed over to the Rendezvous module by the Waypointnavigator module on the *WPDone* event. The

Rendezvous module then transitions to the ***AtRendezvous*** state. If there is no time left for loitering the Rendezvous module transitions to the ***Idle*** state by outputting the event *RendezvousDone*. If there is enough time left the AUV loiters and the sequence is as mentioned in the previous paragraph by passing control to the loiter module. In order to terminate the mission anytime there is a transition from each of the states to the ***Idle*** state on the event *Abort*.

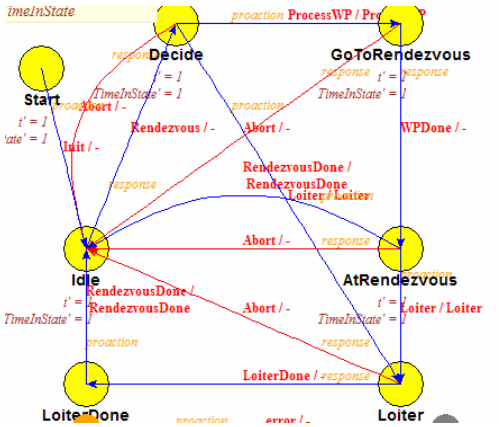


Figure 16: FSM of Rendezvous module in TEJA

2.5.3.8 DeviceCommander

Figure 17 represents the hybrid model of the DeviceCommander. The DeviceCommander sets different devices like the mast. The DeviceCommander transitions to ***SetCommand*** state on receiving the event *SetDevice*. At ***SetCommand*** state the DeviceCommander keeps updating the present condition of the device being set. After the device is set the DeviceCommander transitions to the ***Idle*** state by outputting the event *DeviceDone*. If the device is not set within a specified time (required for a timed mission) a *TimeOut* event occurs and the DeviceCommander transitions to the ***Idle*** state by outputting the *Abort* event. To terminate the mission at anytime if needed there is a transition from any state to the ***Idle*** state on the *Abort* event.

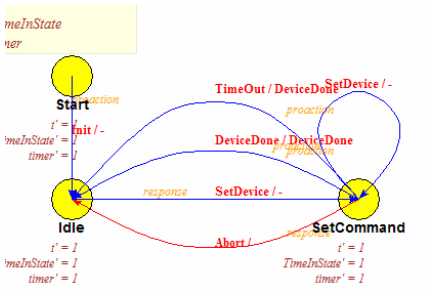


Figure 17: FSM for DeviceCommander in TEJA

2.5.3.9 PayloadDelivery

Figure 18 shows the hybrid representation of the Payload module. This module is used to deliver payload at desired locations. The initial state is ***Start*** state and on the local event *Init* the Payload module transitions to the ***Idle*** state. On receiving the event *ProcessPayload* the Payload module transitions to the ***Run*** state. Then the Payload module passes control to the Waypointnavigator module to go the destined location by outputting the *ProcessWP* event and transitioning to the ***GoToPoint*** state. On reaching the desired point the control is passed to the Payload module by the Waypointnavigator. The Payload module transitions to the ***Deliver*** state either on receiving the event *WPDone* from the Waypoitnnavigator or on receiving the *DeliverPayload* event and outputting the *Abort* event (if the payload needs to be delivered at the present location and mission aborted).

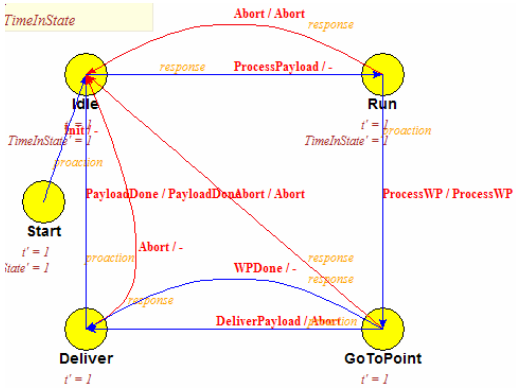


Figure 18: FSM of Payload module in TEJA

2.5.3.10 Loiter

Figure 19 shows the hybrid representation of the Loiter module. The initial state is ***Start*** state. The Loiter module transitions to the ***Idle*** state on the local event *Init*. On receiving the *Loiter* event the Loiter module transitions to the ***ChooseLoiterMode.*** If the loiter mode is not specified the Loiter module transitions to the ***Idle*** state by outputting the *LoiterDone* event. If the loitering mode specified is Hover the Loiter module goes to the ***Hover*** state on the local event *Loiter*. The Loiter module then passes control to the Steer module by outputting the *Steer* event and transitioning to the ***GoToLoiterPt***. If the loiter mode is circle then the Loiter module transitions to the ***Circle*** state on the local event *Loiter*. Then control is passed over to the Steer module to circle the desired location and the Loiter module transitions to the ***GoToCircleWP***. When the distance to point becomes less than a specified number loiter operation is aborted by transitioning to the ***Circle*** state on the event *WPDone* by outputting the *Abort* event. At ***GoToCircleWP*** if the time for circular loitering is over a *TimeOut* event occurs and the loiter module transitions to the ***StopCircle*** state. Then the Loiter module passes control to the Steer module by outputting the *Steer* event to go to the desired location. At ***GoToLoiterPt*** every 2 seconds the Loiter module updates the present location until the desired location is reached. The Loiter module then transitions to the ***ReportTO*** state either on *TimeOut* event for a timed mission or on *WPDone* event on reaching the desired location. Finally the Loiter module ends loiter mission by transitioning to the ***Idle*** state by outputting the *LoiterDone* event.

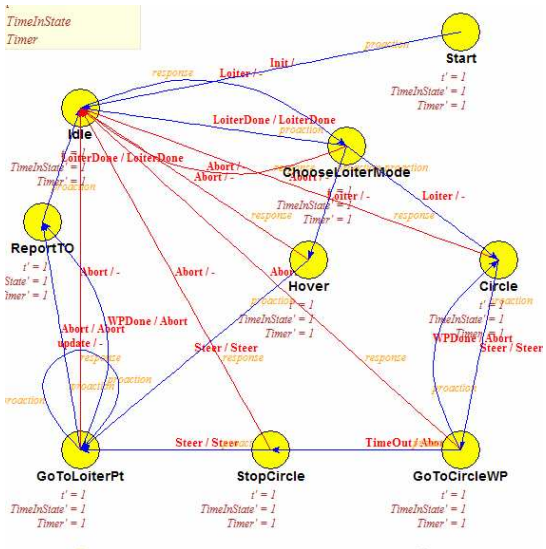


Figure 19: FSM of Loiter module in TEJA

2.5.3.11 Steering

Figure 20 shows the hybrid representation of the Steering module. The Steering module is used to steer the vehicle to the desired location. The initial state is the ***Start*** state. On the local event *Init* the Steering module transitions to the ***Idle*** state. The Steering module transitions to the ***SteerToPoint*** state on receiving the *Steer* event. At ***SteerToPoint*** the Steering module keeps updating the present location to find out whether the desired location is reached or not. When the AUV reaches the desired location the Steering module transition to the ***Idle*** state on the event *SteeringDone*. To terminate the mission when needed the Steering module reacts to the *Abort* event to end the steering operation.

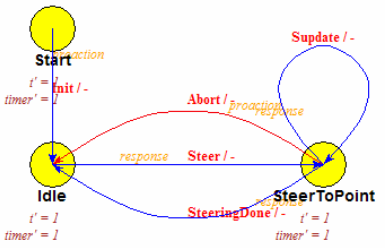


Figure 20: Steering behavior controller in TEJA

3 Bottom up verification approach

Many of these applications are *safety critical* and require guarantees of safe operation together with correctness or progress. Informally, safety refers to the requirement that *nothing bad should ever occur* (such as the underwater vehicle should never enter a shallow region and collide with it) whereas correctness requires that *something good should eventually happen* (such as the underwater vehicle should eventually accomplish the mission task). Consequently, much research focuses on formal modeling and verification/synthesis of hybrid systems, including [19], [20], [10], [21], [28], [70], [24], [30].

In section 3.1 we discuss modeling and verification of hybrid systems. In section 3.2 we explain our own approach. In section 3.3 we discuss our algorithm and demonstrate its working, and finally we conclude and propose future work in section 3.4.

3.1 Verification of Hybrid systems

Formal verification is defined as an organized exhaustive method to check the correctness of a system. Verification of a hybrid system involves complex issues related to both the discrete and continuous domain and proper formulation of correctness. Several techniques and tools have been developed for hybrid system verification.

Two different approaches to formal verification are algorithmic and deductive verification. For algorithmic verification to be possible the corresponding algorithms should terminate. This is not the case for general hybrid systems for the simplest of problems such as reachability. Algorithmic verification is thus restricted to few classes of hybrid systems possessing a finite abstraction [24]. Deductive verification is the method of verification to reduce the verification task into establishment of premises in a set of proof rules. Deductive approach is more powerful and least restrictive as it can be used for verification of anything. The disadvantage of deductive verification is that it needs more human interaction and requires more expertise.

Model checking developed independently by Clarke and Emerson, and by Queille and Sifakis in 1980's, is an algorithmic approach to verification, in which a model of the system together with the specification/query defining the desired requirement/property is provided as the inputs to a model checker, which checks whether the requirements hold

for all the behaviors of the system. For model checking, different ways to model a system are as Discrete Automaton, Timed Automaton, and Hybrid Automaton. The queries can be a formula in LTL (Linear Time Temporal Logics), CTL (Computation Tree Logic), TCTL (extends CTL with timing constraints) [76], ICTL (Integrator Computation Tree Logic) [76], or another temporal logic to suit the application. There are several tools available implementing the model checking methods. Discrete Automaton and logics like CTL or LTL are supported in SMV and SPIN. Timed automata with real time logics are supported in UPPAAL [23], KRONOS [77], and extensions of SPIN [36]. For linear hybrid automata HyTech checks reachability of given set of states. d/dt is another tool also for reachability analysis of a hybrid systems with linear differential inclusions.

Theorem proving is a deductive approach. In order to prove whether a property is satisfied by the implemented system, a proof is created based on mathematical reasoning. Several theorem proving systems have been developed such as, Boyer-Moore (First order logic) [71], HOL (Higher order logic) [79], PVS (higher order logic) [80], Lambda (higher order logic) [81], TPS (first- and higher-order logic) [82] to semi-automate or to make theorem proving interactive. Theorem proving has its limitation that its complete automation is generally impossible, and can mostly be used by the experts.

Our work involves use of algorithmic methods for verification of hybrid system that are hierarchically structured, in a certain way that helps reduce the computational complexity of verification.

3.2 Bottom up approach to verification

Our mission control architecture is designed with semi-automatic verification in mind. All the levels and modules that make up the hierarchy are interacting controlled hybrid system described in chapter 2. The tool used to implement the hierarchy allows the conversion into a format that is readable by available verification tools such as HyTech [79] and Uppaal [23]as discussed in 2.6. Interactions among modules occur through event synchronizations and sharing of data. The continuous dynamics within individual automata only uses clocks. While designed within existing mission control architectures have been deployed successfully, none of them were designed to be used with modern

verification tools and techniques without considerable overhead. The architecture is discussed in details in chapter 2.

A bottom up approach to verification aims to simplify the verification of a complex, hierarchically structured system, initiating verification from the lowest level. The lowest level is verified as a single entity interacting with an "abstract" model of its environment comprising of the lateral and higher level modules. Recursively, the next higher level subsystems are verified, and added to the lower levels that have all been verified. This recursive addition of level continues until all the levels are verified. Each level constitutes of several controller modules (behavioral or operational or coordinator in our case). Similar kind of approach was used in discrete domain for controller synthesis in manufacturing systems [83].

The sequence of steps each controller module executes depends on the mission or other types of orders and responses received from modules at higher or lateral or lower levels. The issues involved in this method of verification are:

1. identifying discrete logic
2. identifying properties to be checked
3. modeling the abstract model of the environment

We can verify the discrete logic separately from the continuous dynamics associated with each step. Logical correctness of the whole system requires checking the discrete logical sequence of steps executed in response to orders and responses by the interacting modules based on the invariants, guards, events, reset values and the state transitions, computed with each discrete logical step.

In this work we are concerned with correctness of logic part that ensures correct execution of discrete steps, verified algorithmically using the model checker UPPAL. The correctness of output assignments involving functions associated with each discrete step is not treated here, but can be achieved using a theorem prover such as PVS. The properties to be checked for logical correctness have been formulated based on the mission/task the system needs to execute. When verifying a certain module, the other interacting modules which constitute its environment are optionally abstracted as simpler modules. The "commanding" modules are abstracted as *drivers*, whereas the

"commanded" modules are abstracted as *stubs*. The abstraction preserve the essential features needed for adequate interaction with the module being verified.

Following the above bottom-up approach, the process of checking progress for a hierarchical interacting hybrid system, $\mathcal{H} = \mathcal{H}^1_1 \| \ldots \mathcal{H}^j_i \ldots \| \mathcal{H}^m_n$, where $\mathcal{H}^j_i = \{Q^j_i, \Sigma^j_i, U^j_i, Y^j_i, F^j_i, H^j_i, I^j_i, E^j_i, G^j_i, R^j_i\}$, is the hybrid automaton model of the jth module ($j=1 \ldots m_i$) on the ith level ($i=1 \ldots n$), the following algorithm describes the verification approach.

For $i = 1$ *to* n

For $j = 1$ *to* m_i

- Select subsystem $\mathcal{H}^j_i$ for verification
- Find all subsystems $\mathcal{H}^l_k$, $k=1$ *to* n, $l=1$ *to* m_k, $(k,l) \neq (i,j)$ that interact with subsystem $\mathcal{H}^j_i$
- Abstract all subsystems $\mathcal{H}^l_k$, for k,l as found above, whose internal states are not relevant to verification, as *drivers* or *stubs*, and replace the original subsystems with the abstracted subsystems
- Compose the system as $\mathcal{H}' = \mathcal{H}^j_i \| \mathcal{H}^l_k$ for k,l as found above
- Formulate queries using temporal logic formulas based on the progress requirements of the system, check queries on composed system $\mathcal{H}'$, and decide on progress
- Correct any correctness problems found for module $\mathcal{H}^j_i$

Next j

Next i

3.2.1 Properties satisfied by the algorithm

Property 3.1: *No hybrid module is left without consideration for verification*

If $(\forall i(1 \le i \le n) \forall j(1 \le j \le m_i) \exists H \mid H^j_i = True) \,\&\, (\Delta j = True, \Delta i = True)$ then we can say that none of the hybrid modules are left without verification, where i represents the i^{th} level, j represents the j^{th} module, Δi indicates going to next level by incrementing i and Δj indicates selecting the next module by incrementing j.

The statement above states that for each of the levels and each of the subsystems at that level there exist a subsystem which is selected for verification and there is a method to select the next subsystem at a level and to select the next higher level within the hierarchical structure. This statement indicates that all the subsystems are considered for verification one by one. Considering the statements from the algorithm as shown below.

For i = 1 to n can be expressed as $(\forall i(1 \le i \le n))$ which indicates that the algorithm starts by considering the lowest level *i = 1*. --- 1

For j = 1 to m_i can be expressed as $(\forall j(1 \le j \le m_i))$ which indicates that the algorithm looks to select for a subsystem at the level *i*, where m_i indicates the total number of susbsystems at level *i*. --- 2

The two for loops one after the other as shown in the algorithm can be expressed as $(\forall i(1 \le i \le n)\forall j(1 \le j \le m_i))$ which indicates that the algorithm first selects a level and then selects a subsystem at that level. --- 3

The statement Select subsystem $\mathcal{H}_i^j$ for verification within the algorithm can be expressed as $(\forall i(1 \le i \le n)\forall j(1 \le j \le m_i)\exists H \mid H_i^j = True)$ which indicates that at level *i* a subsytem is selected. --- 4

The updation statement Next *j* indicates that the algorithm next searches for the subsystem at the present level *i* which can be expressed as $(\forall i(1 \le i \le n)\forall j(1 \le j \le m_i)\exists H \mid H_i^j = True) \,\&\, (\Delta j = True)$. --- 5

The updation statement Next *i* indicates that the algorithm next searches to select the next higher level which can be expressed as $(\forall i(1 \le i \le n)\forall j(1 \le j \le m_i)\exists H \mid H_i^j = True) \,\&\, (\Delta i = True)$. --- 6

The updation statements one after the other as shown in the algorithm can be expressed as $(\forall i(1 \le i \le n)\forall j(1 \le j \le m_i)\exists H \mid H_i^j = True) \,\&\, (\Delta j = True, \Delta i = True)$. --- 7

Equations 1-7 show that each and every susbsystem is selected one by one so the ***Property 3.1*** holds.

Property 3.2: *The subsystem alongwith the abstracted environment interacts the same way as without abstraction.*

Assumption: There is no dependency of the abstracted environment on any other module other than the concerned module for execution of actions.

If $(\forall i(1 \le i \le n)\forall j(1 \le j \le m_i)\exists H \mid H_i^j \parallel H_k^l \ne Deadlock) \,\&\, (\Delta j = True, \Delta i = True)$ holds the above property holds

Equations 1-4 show that a subsystem is selected for verification. According to the algorithm next we need to *Abstract all subsystems* $\mathcal{H}^{l}_{k}$*, for k,l as found above, whose internal states are not relevant to verification, as drivers or stubs, and replace the original subsystems with the abstracted subsystems*. The abstracted subsystem $\mathcal{H}^{l}_{k}$ contains guard conditions g_e, and syhnchronization events σ, using which the selected subsystem $\mathcal{H}^{j}_{i}$ interacts with its environment.

The next step is to *Compose the system as* $\mathcal{H}' = \mathcal{H}^{j}_{i} \| \mathcal{H}^{l}_{k}$ *for k,l as found above.* The above abstraction method results in no deadlock as the events and guards on which the selected subsystem reacts to are still present. This can be expressed as

$(\forall i(1 \le i \le n)\forall j(1 \le j \le m_i)\exists H \mid H_i^j \parallel H_k^l \ne Deadlock)$ --- 8

Equations 5-7 prove the next part of the equation as expressed in the proof. Thus

$(\forall i(1 \le i \le n)\forall j(1 \le j \le m_i)\exists H \mid H_i^j \parallel H_k^l \ne Deadlock) \,\&\, (\Delta j = True, \Delta i = True)$ holds

which indicates that ***Property 3.2*** holds.

Property 3.3: *Properties satisfied by each of the hybrid susbsystems are verified*

If $(\forall i(1 \le i \le n)\forall j(1 \le j \le m_i)(\exists H \exists \Phi \mid (H_i^j \parallel H_k^l) \models \Phi) \,\&\, (\Delta j = True, \Delta i = True)$ ***Property 3.3*** holds.

Equations 1-8 show that a subsystem is selected at a level, its environment is abstracted and finally the subsystem interacts with the abstracted environment without any deadlocks. The composed subsystem obtained is a semantical model H and the property to be checked is a logical formula Φ. Here the property Φ is expressed as a temporal logic formula. The temporal logic formula checks the satisfaction of the property Φ based on the reachablity of the state or path at which the property is satisfied. Thus it proves that using the method of model checking we can verify the satisfaction of properties by the hybrid subsystem. Thus

$(\forall i(1 \le i \le n)\forall j(1 \le j \le m_i)(\exists H \exists \Phi \mid (H_i^j \parallel H_k^l) \models \Phi) \,\&\, (\Delta j = True, \Delta i = True)$ --- 9

holds which proves the truth of ***Property 3.3***.

Property 3.4: *The logical correctenes of all the hybrid susbsytems involved in a hierarchical architecture can be verified using this algorithm if*

(a) *No hybrid module is left without consideration for verification*

(b) *The subsystem alongwith the abstracted environment interacts the same way as without abstraction*

(c) *Properties satisfied by each of the hybrid susbsystems are verified*

Equations 1-9 validate ***Property 3.4***

3.3 Uppaal: Tool for Verification

As described, the hybrid mission controller has been designed using Teja NP. Teja is a tool that facilitates the graphical design of interacting hybrid automata and includes real-time code generation utilities. Teja, however, does not contain functionality for formal verification; thus we use Uppaal for verification. In order to facilitate rapid (re)design and verification, a Teja to Uppaal converter, called *dem2xml*, was created at ARL PSU that converts a hybrid (timed) autonomous system description in Teja to an Uppaal system description.

Uppaal is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata, extended with data types. The tool is designed to verify systems that can be modelled as networks of timed automata extended with integer variables, structured data types, and channel synchronisation. It contains two parts a graphical user interface and a model check engine. The user interface is implemented in java and executed on users work station. The model-checker Uppaal is based on the theory of timed automata and its modelling language offers additional features such as bounded integer variables and urgency. A timed automaton is a finite-state machine extended with clock variables. It uses a dense-time model where a clock variable evaluates to a real number. The query language of Uppaal, used to specify properties to be checked, is a subset of CTL (computation tree logic). The query language consists of a path formula and state formula. State formula describes individual states, whereas path formulae quantify over paths or traces of the model. Path formula can be classified into reachablity (given state formula will be satisfied by a given state), safety (nothing bad will ever happen) and liveness (something will eventually happen). The tool consists of an editor, a simulator and a verifier. The editor is divided into two parts: a tree pane to access the different templates and declarations and a drawing canvas/text editor. A timed automaton is modeled in the drawing canvas. A state of the system is defined by

the locations of all automata, the clock constraints, and the values of the discrete variables. A timed automaton contains locations and the behavior of timed automata is modeled by the transitions between the locations. A system is defined as a network of timed automata, called processes in the tool, put in parallel. A process is instantiated from a parameterized template. The simulator can be used in three ways: the user can run the system manually and choose which transitions to take, the random mode can be toggled to let the system run on its own, or the user can go through a trace (saved or imported from the verifier) to see how certain states are reachable. The Verifier is used to give queries to be verified. Properties are selectable in the Overview list. The verifier checks for simple invariants and reachability properties i.e. a certain state is reachable or not. The queries are in the form of temporal logic formulas. The user may model-check one or several properties, insert or remove properties, and toggle the view to see the properties or the comments in the list. Satisfied properties are marked green and violated ones red.

For running large verification tasks, it is often cumbersome to execute these from inside the GUI. For such situations, the stand-alone command line verifier called verifyta can be used. It also makes it easy to run the verification on a remote UNIX machine with memory to spare.

Two, and only two, Uppaal subsystems may synchronize on two enabled edges over a normal channel if one edge is *commanding* and one edge is *accepting*. *Any one* Uppaal subsystem with an enabled edge may synchronize with the commanding subsystem, and if no synchronizing edge is available, no transition will take place; whereas in Teja, the transition will take place in the commanding subsystem regardless of how many systems, including zero, are synchronizing on the event.

To overcome this problem, we declare all channels in Uppaal as broadcast channels. Zero, one, or multiple Uppaal subsystems may synchronize on a single event over a broadcast channel. We are however restricted in that a certain subsystem, not listed to receive an event in the Teja event dependency table, may still synchronize on that event in Uppaal. This restriction must be overcome by examining the Teja event dependency table during Uppaal verification. These are some of the issues to be noted between Teja and Uppaal. Now that we know how the modules are modeled using Teja lets look into the verification of the modules using Uppaal next.

3.4 Illustration of Logical correctness – Survey AUV

We start with the **lowest level** (*i=1*), and pick the **Steering** module first as it receives orders from the others, and only responds to those orders. We develop abstract models of the *commanding* and *commanded* environment, called drivers and stubs. The steering module is shown in Figure 21, whereas the abstraction of commanding environment the driver module is shown in Figure 22. There is no module that the steering module commands.

3.4.1 Verification of Steering module

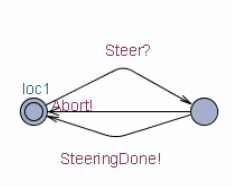


Figure 21: Steering module in UPPAAL

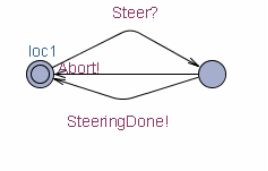


Figure 22: Driver for steering module in UPPAAL

Queries

The following queries were formulated as temporal logic formulae in order to perform verification of logical correcteness of the steering module.

E[] Steering_P.Idle_ds

/*Eventually in future there always is a path, which goes to the final state (here Idle_ds)*/

A[] Steering_P.SteerToPoint_ds imply Steering_P.timer<=2

/*The steering module always updates the present location of the AUV every 2 seconds when at state SteerToPoint_ds */

A<> Steering_P.SteerToPoint_ds imply Steering_P.Idle_ds

/* Always eventually the vehicle is steered to the desired point */

3.4.2 Verification of Loiter module

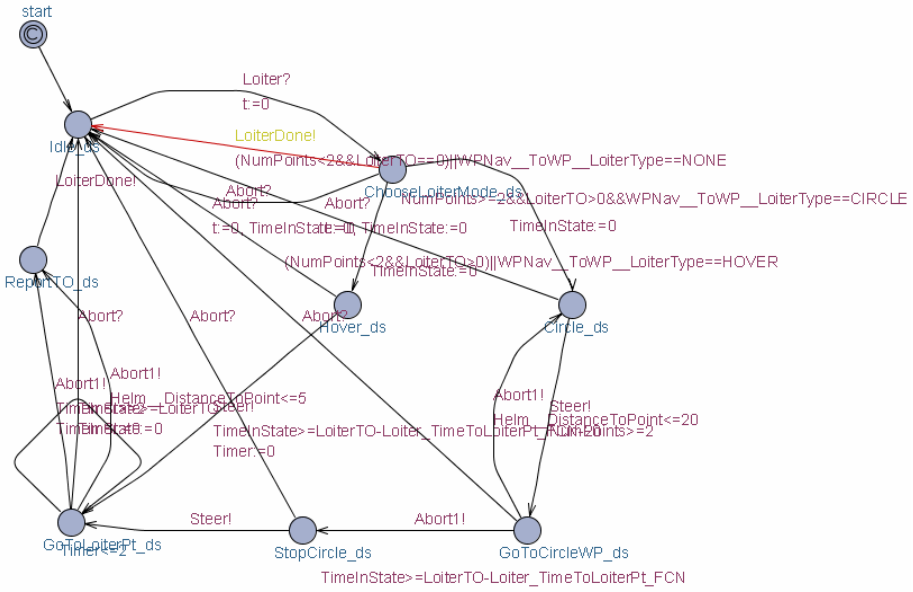


Figure 23: Loiter module module in UPPAAL

The next module selected is the Loiter module at level 1, then its environment is abstracted. The Loiter module is shown in Figure 23, the abstracted commanding environment is shown in Figure 24 and the abstracted commanded environment is shown in Figure 25.

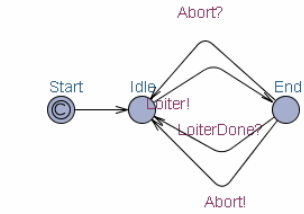


Figure 24: Driver for loiter module module in UPPAAL

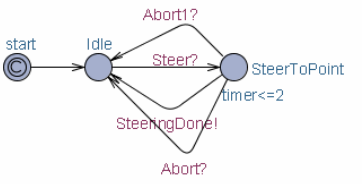


Figure 25: Stub for loiter module module in UPPAAL

Queries formulated as temporal formulas are as shown below.

A<> Loiter_P.Idle_ds

/* All paths eventually lead to the final or end state (here Idle_ds), indicating progress i.e. order is completed as final state is reached or it might be that the mission is aborted but that too signifies progress indicating the detection of failure and no deadlock. */

A<> Loiter_P.ChooseLoiterMode_ds imply (Loiter_P.Hover_ds||Loiter_P.Circle_ds)
/* All paths from the state ChooseLoiterMode eventually either leads to hovering or circling (indicating progress as it should select a mode to loiter). */

A[] Loiter_P.Circle_ds imply (Loiter_P.LoiterTO>0 && Loiter_P.NumPoints>2)&& WPNav__ToWP__LoiterType==Loiter_P.CIRCLE
/* For all paths if "Circling" is selected the guard conditions are LoiterTO>0 and NumPoints>2 (this indicates that the guard conditions required for circling like number of points are taken care of) Changes: numpoints>2 and LoiterTO>0 in global declaration */

A[] Loiter_P.Hover_ds imply (Loiter_P.LoiterTO>0 && Loiter_P.NumPoints>2)|| WPNav__ToWP__LoiterType==Loiter_P.HOVER
/* For all paths always if "Hovering" is selected then the required guard conditions are satisfied. Changes: NumPoints<2 and LoiterTO>0 */

A[] Loiter_P.Circle_ds imply Loiter_P.NumPoints>2
/* For all paths always circling means number of points >2 Changes: NumPoints>=2 */

E<> not Loiter_P.NumPoints>2 and Loiter_P.Hover_ds
/* This statement proves that there doesnt exist a path where eventually NumPoints>2 hold after Hover mode */

E<> (Loiter_P.NumPoints<2 && WPNav__ToWP__LoiterType==Loiter_P.HOVER) imply Loiter_P.Hover_ds
/* Is it possible to reach Hover mode with all parameters for HOVERING mode */

E<> (Loiter_P.NumPoints>2 && WPNav__ToWP__LoiterType==Loiter_P.CIRCLE) and Loiter_P.Hover_ds

/* Is it possible to reach Hover mode with all parameters for Circling mode */

E<> Loiter_P.NumPoints>2 imply Loiter_P.GoToCircleWP_ds

/* Does there exist a path where NumPoints>2 leads to GoToCircleWP_ds (indicating progress of Circle type order being executed correctly) */

E<> (Loiter_P.NumPoints>2 && WPNav__ToWP__LoiterType==Loiter_P.CIRCLE) imply Loiter_P.Circle_ds

/* Is it possible to reach CIRCLE mode with circling mode type input (indicating that logic of guards for progress of executing orders correctly)*/

E<> (Loiter_P.NumPoints<2 && WPNav__ToWP__LoiterType==Loiter_P.HOVER) and Loiter_P.Circle_ds

/* Is it possible to reach CIRCLE mode with hovering mode type input (indicating that guards for progress are given correctly)*/

E<> Loiter_P.TimeInState<=Loiter_P.LoiterTO-Loiter_P.Loiter_TimeToLoiterPt_FCN imply not Loiter_P.StopCircle_ds

/*Is it possible to reach StopCircle_ds when the guard condition leading to that state is satisfied */

E<> (Loiter_P.GoToCircleWP_ds and Helm__DistanceToPoint<=20) imply Loiter_P.Circle_ds

/* Check correct execution: Is it possible to reach to Circle_ds when at GoToCircleWP_ds and guard Helm__DistanceToPoint<=20 is satisfied */

E<> (Loiter_P.Hover_ds || Loiter_P.Circle_ds) imply Loiter_P.GoToLoiterPt_ds

/*Is it possible to reach the GoToLoiterPt_ds state after choosing the mode of loitering */

A<> Loiter_P.ChooseLoiterMode_ds imply Loiter_P.Idle_ds

/* All paths eventually lead to final state from the present state indicating either successful completion or termination of a mission*/

A<> Loiter_P.Hover_ds imply Loiter_P.Idle_ds

/* All paths eventually lead to final state from the present state */

A<> Loiter_P.Circle_ds imply Loiter_P.Idle_ds

/* All paths eventually lead to final state from the present state */

A<> Loiter_P.GoToCircleWP_ds imply Loiter_P.Idle_ds

/*All paths eventually lead to final state from the present state*/

A<> Loiter_P.StopCircle_ds imply Loiter_P.Idle_ds

/*All paths eventually lead to final state from the present state*/

A<> Loiter_P.GoToLoiterPt_ds imply Loiter_P.Idle_ds

/*All paths eventually lead to final state from the present state */

A<> Loiter_P.ReportTO_ds imply Loiter_P.Idle_ds

/*All paths eventually lead to final state from the present state*/

E<> Loiter_P. GoToLoiterPt_ds and Loiter_P.Timer<=2

/*Is the information regarding reaching the loiter point updated every 2 seconds*/

3.4.3 Verification of GPSFixer module

All the subsystems at level 1 are now verified. So the value of *i* changes to the next level which is 2. The next module selected is the GPSFixer subsystem at level 2. The environment for GPSFixer susbsystem is now abstracted. The GPSFixer subsystem is shown inFigure 26, the abstracted commanding environment is shown in Figure 27 and the abstracted commanded environment is shown in Figure 28.

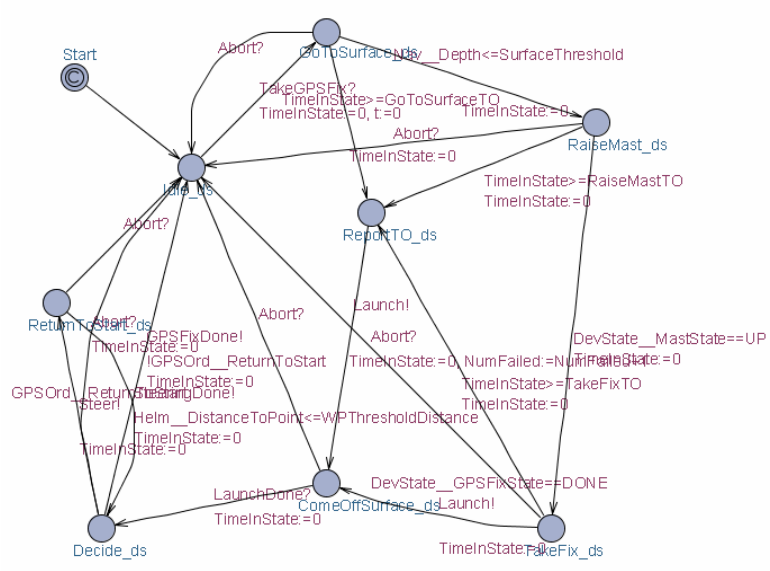


Figure 26: GPSFixer module in UPPAAL

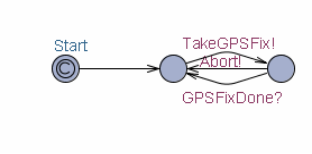


Figure 27: Driver for GPSFixer module in UPPAAL

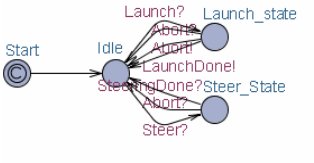


Figure 28: Stub for GPSFixer module in UPPAAL

Queries:

A<> GPSFixer_P.Idle_ds

/* All paths eventually lead to the final state (All paths not always lead to final state as ReportTO_ds doesnt connect to the final state but it eventually goes to the final state).*/

E<> (GPSFixer_P.GoToSurface_ds and Nav__Depth<=GPSFixer_P.SurfaceThreshold) imply GPSFixer_P.RaiseMast_ds

/*Is it possible to reach the next state (RaiseMast_ds) when its at present state GoToSurface_ds and guard condition is Nav__Depth<=SurfaceThreshold */

E<> DevState__MastState==GPSFixer_P.UP and GPSFixer_P.RaiseMast_ds imply GPSFixer_P.TakeFix_ds

/*Is it possible to reach the next state (TakeFix_ds) when its at present state RaiseMast_ds and the guard condition checking whether the mast is raised is satisfied */

E<> DevState__GPSFixState==GPSFixer_P.DONE and GPSFixer_P.TakeFix_ds imply GPSFixer_P.ComeOffSurface_ds

/*Is it possible to reach the next state (ComeOffSurface_ds) when its at present state TakeFix_ds and the guard condition checking whether the GPSFix is done is satisfied*/

A[] not deadlock

/*Does there exist a deadlock*/

E<> GPSFixer_P.ComeOffSurface_ds imply GPSFixer_P.Decide_ds || GPSFixer_P.Idle_ds

/*Is it possible to reach the next state (Decide_ds or Idle_ds) when its at present state ComeOffSurface_ds */

E<> GPSFixer_P.Decide_ds and GPSOrd__ReturnToStart and GPSFixer_P.ReturnToStart_ds

/* Is it possible to reach the next state (ReturnToStart_ds) when its at present state Decide_ds and the guard conditon checking whether to return to start stae is satisfied*/

E<> GPSFixer_P.Decide_ds and !GPSOrd__ReturnToStart imply GPSFixer_P.Idle_ds

/* Is it possible to reach the next state (Idle_ds) when its at present state Decide_ds and the guard conditon checking whether not to return to start state is satisfied */

E<> GPSFixer_P.Decide_ds and GPSOrd__ReturnToStart and Steering_P.SteerToPoint_ds

/* Is it possible to reach SteerToPoint_ds in Steering when the guard condition is satisfied when in the Decide_ds state in GPSFixer */

E<> GPSFixer_P.Decide_ds and !GPSOrd__ReturnToStart and Steering_P.SteerToPoint_ds

/* is it possible to reach from Decide_ds in GPSFixer to Steering (SteerToPoint_ds) when the guard condition is not satisfied */

E<> GPSFixer_P.Decide_ds and !GPSOrd__ReturnToStart and Steering_P.Idle_ds

E<> GPSFixer_P.TakeFix_ds and DevState__GPSFixState==GPSFixer_P.DONE and Launcher_P.RetractMast_ds

/*is it possible to reach from TakeFix_ds in GPSFixer to Launcher (RetractMast_ds) when the guard condition is not satisfied and the synchronous event occurs */

3.4.4 Verification of Waypointnavigator module

The value of level *i* remains 2. The next module selected is the Waypointnavigator subsystem at level 2. The environment for Waypointnavigator susbsystem is now abstracted. The Waypointnavigator subsystem is shown in Figure 29, the abstracted commanding environment is shown in Figure 30 and the abstracted commanded environment is shown in Figure 31.

Queries for the verification of Waypointnavigator is shown belo.

A[] not deadlock

/*Does there exist a deadlock*/

A<> WaypointNavigator_P.Idle_ds

/* All paths eventually lead to the final state (inidicating progress or diagnosis of failure if aborted) */

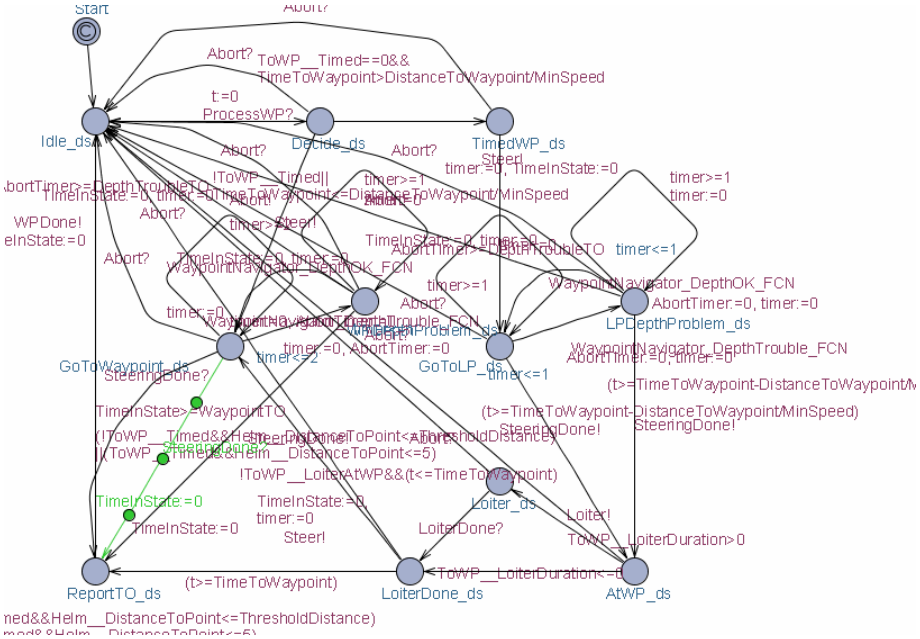


Figure 29: Waypointnavigator module in UPPAAL

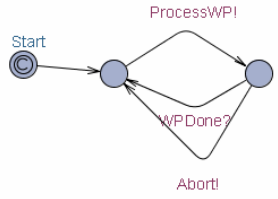


Figure 30: Driver for Waypointnavigator module in UPPAAL

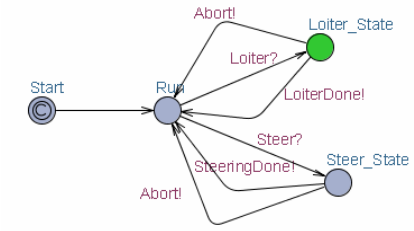


Figure 31: Stub for Waypointnavigator module in UPPAAL

E<> ToWP__Timed && WaypointNavigator_P.TimeToWaypoint > WaypointNavigator_P.DistanceToWaypoint / WaypointNavigator_P.MinSpeed and WaypointNavigator_P.TimedWP_ds

/* Is it possible to do a timed waypoint if the guard conditions are satisfied (Yes indicates correct implementation) Change: TimeToWaypoint DistanceToWaypoint MinSpeed to check the various situations with the guard condition */

E[] (WaypointNavigator_P.TimeToWaypoint > WaypointNavigator_P.DistanceToWaypoint /WaypointNavigator_P.MinSpeed) imply \ WaypointNavigator_P.TimedWP_ds

/* Whenever the guard signifying time is satisfied does it do timed waypoint */

E<> !ToWP__Timed || WaypointNavigator_P.TimeToWaypoint <= WaypointNavigator_P.DistanceToWaypoint/WaypointNavigator_P.MinSpeed and WaypointNavigator_P.GoToWaypoint_ds

/* Is it possible to do an untimed waypoint if the guard conditions are satisfied (Yes indicates correct implementation) Change: TimeToWaypoint DistanceToWaypoint MinSpeed to check the various situations with the guard condition */

E[] (WaypointNavigator_P.TimeToWaypoint < WaypointNavigator_P.DistanceToWaypoint / WaypointNavigator_P.MinSpeed) imply WaypointNavigator_P.GoToWaypoint_ds

/*Is it possible to do untimed waypoint when time to reach the point by the vehicle is more than desired time*/

WaypointNavigator_P.TimedWP_ds --> WaypointNavigator_P.GoToLP_ds

/*Is it possible to reach go to loiter point state once timed waypoint is started (as the next step is to go to loiter point indicating progress)*/

A<> (WaypointNavigator_P.GoToWaypoint_ds and WaypointNavigator_P.WaypointNavigator_DepthTrouble_FCN) imply WaypointNavigator_P.WPDepthProblem_ds

/* For all paths at state GoToWaypoint_ds if depth trouble occurs it goes to rectify it.*/

E<> (WaypointNavigator_P.WaypointNavigator_DepthOK_FCN and WaypointNavigator_P.WaypointNavigator_DepthTrouble_FCN) imply WaypointNavigator_P.GoToWaypoint_ds

/*Is it possible to get depth rectified and move on to normal state*/

E[] WaypointNavigator_P.GoToWaypoint_ds imply Stub.Steer_State

/*There exists a path where always steering needs to be done whenever waypoint needs to go to a point*/

E<> *(!ToWP__Timed&&Helm__DistanceToPoint<= WaypointNavigator_P.ThresholdDistance)* || (ToWP__Timed&&Helm__DistanceToPoint <= 5) imply Stub.Run

/* The guard conditions being satisfied steering is done, the point is reached*/

E<> *(WaypointNavigator_P.GoToWaypoint_ds* || *WaypointNavigator_P.WPDepthProblem_ds)&&(!ToWP__Timed&&Helm__DistanceTo Point* <= *WaypointNavigator_P.ThresholdDistance)||(ToWP__Timed&&Helm__DistanceToPoint<=5) imply Stub.Run && WaypointNavigator_P.ReportTO_ds*

/* Is it possible to go to the Idle state in steering and ReportTO state in Waypoint from go to waypoint or depth problem rectifying state in waypoint when the guard conditions are satisfied*/

E<> WaypointNavigator_P.GoToLP_ds imply Stub.Steer_State

/* Is it possible to steer to desired point when doing timed waypoint */

E<> *(WaypointNavigator_P.GoToLP_ds and WaypointNavigator_P.WaypointNavigator_DepthTrouble_FCN) imply WaypointNavigator_P.LPDepthProblem_ds*

/* Is it possible to go to depth correction state when the guard indicating depth trouble is TRUE (yes indicates correct performance) */

E<> WaypointNavigator_P.LPDepthProblem_ds and WaypointNavigator_P.timer>=2

/*Is it possible to be at state LPDepthProblem_ds with timer greater than 2 (if yes indicating wrong model) couldnt find result as taking lot of time need to check for long */

E<> WaypointNavigator_P.LPDepthProblem_ds and WaypointNavigator_P.timer<=1

/* Is it possible to be at state LPDepthProblem_ds with timer less than equal to 1 (if yes indicating correct model) couldnt find result as taking lot of time need to check for long */

E<> WaypointNavigator_P.GoToLP_ds and WaypointNavigator_P.timer>=2

/* Is it possible to be at state GoToLP_ds with timer greater than 2 (if yes indicating wrong model) couldnt find result as taking lot of time need to check for long*/

E<> WaypointNavigator_P.GoToLP_ds and WaypointNavigator_P.timer<=1

/* Is it possible to be at state GoToLP_ds with timer less than equal to 1 */

E<> WaypointNavigator_P.GoToWaypoint_ds and WaypointNavigator_P.timer>=3

/* Is it possible to be at state GoToWaypoint_ds with timer greater than 3 */

E<> WaypointNavigator_P.GoToWaypoint_ds and WaypointNavigator_P.timer<=2

/*Is it possible to be at state GoToWaypoint_ds with timer less than equal to 2 */

E<> WaypointNavigator_P.WPDepthProblem_ds and WaypointNavigator_P.timer>=2

/*Is it possible to be at state GoToWaypoint_ds with timer greater than 2 */

E<> WaypointNavigator_P.WPDepthProblem_ds and WaypointNavigator_P.timer<=1

/* Is it possible to be at state GoToWaypoint_ds with timer less than equal to 1 */

E<> WaypointNavigator_P.TimedWP_ds imply (WaypointNavigator_P.Loiter_ds || WaypointNavigator_P.LoiterDone_ds)

/*Is it possible to loiter if executing a timed waypoint */

E<> WaypointNavigator_P.AtWP_ds and ToWP__LoiterDuration<=0 imply WaypointNavigator_P.LoiterDone_ds

/* Is it possible to reach the desired region when no time is left for loitering while doing a timed waypoint */

E<> WaypointNavigator_P.AtWP_ds and ToWP__LoiterDuration>0 imply WaypointNavigator_P.Loiter_ds

/* Is it possible to loiter when time is left to reach the desired region while doing a timed waypoint */

E<> WaypointNavigator_P.LoiterDone_ds and (WaypointNavigator_P.t >=WaypointNavigator_P.TimeToWaypoint) imply WaypointNavigator_P.ReportTO_ds

/* Is it possible to reach the data reporting state after loitering is done */

E<> WaypointNavigator_P.ReportTO_ds imply WaypointNavigator_P.Idle_ds

/* Is it possible to reach the final state when the desired point is reached */

3.4.5 Verification of Rendezvous module

The value of level *i* remains 2. The next module selected is the Rendezvous subsystem at level 2. The environment for Rendezvous susbsystem is now abstracted. The Rendezvous subsystem is shown in Figure 32, the abstracted commanding environment is shown in Figure 33 and the abstracted commanded environment is shown in Figure 34.

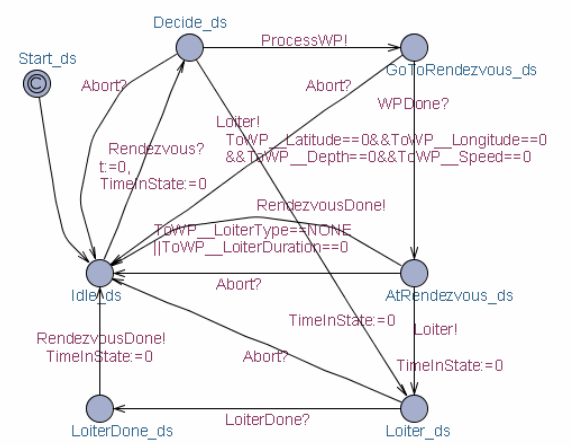


Figure 32: Rendezvous module in UPPAAL

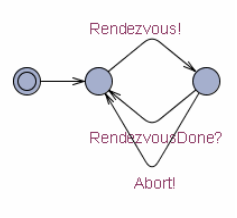


Figure 33: Driver for Rendezvous module in UPPAAL

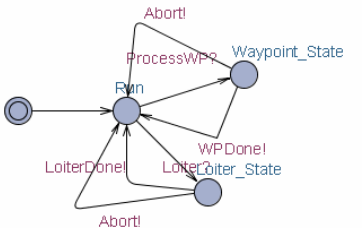


Figure 34: Stub for Rendezvous module in UPPAAL

Queries as temporal logic formulas to verify properties for the Rendezvous susbsystem.

A[] not deadlock

/*Does there exist a deadlock*/

E<> ToWP__Latitude==0&&ToWP__Longitude == 0 && ToWP__Depth == 0 && ToWP__Speed == 0 imply Rendezvous_P.Loiter_ds

/* Is it possible to loiter when rendezvous is not specified*/

E<> ToWP__LoiterType==Rendezvous_P.NONE ||ToWP__LoiterDuration==0 imply Rendezvous_P.Idle_ds

/* Is it possible to be idle when no loiter type is given and no loiter duration is specified*/

E<> Rendezvous_P.GoToRendezvous_ds and Stub.Waypoint_State

/*Is it possible to start waypoint navigation by the rendezvous controller */

E<> Rendezvous_P.AtRendezvous_ds and Stub.Run

/* Is it possible for rendezvous to synchronize with waypointNavigator to process way point navigation */

E<> Rendezvous_P.Loiter_ds and Stub.Loiter_State

/* Is it possible to synchronize Rendezvous with Loiter for loitering*/

E<> Rendezvous_P.LoiterDone_ds and Stub.Run

/* Is it possible to synchronize rendezvous with loiter to get loitering done */

Rendezvous_P.LoiterDone_ds --> Stub.Run

/*Is it possible by the rendezvous controller to execute loiter successfully*/

E<> Rendezvous_P.LoiterDone_ds and Stub.Run

/*Checking on synchronization of rendezvous with loiter*/

3.4.6 Verification of Launcher module

The value of level *i* remains 2. The next module selected is the Launcher subsystem at level 2. The environment for Launcher susbsystem is now abstracted. The Launcher subsystem is shown in Figure 35, and the abstracted commanding environment is shown in Figure 36.

Queries as temporal logic formulas to verify the properties satisfied by the Launcher subsystem.

A[] not deadlock

/*Does there exist a deadlock*/

E<> Launcher_P.Idle_ds imply Launcher_P.RetractMast_ds

/* Is it possible to retract mast using the launcher */

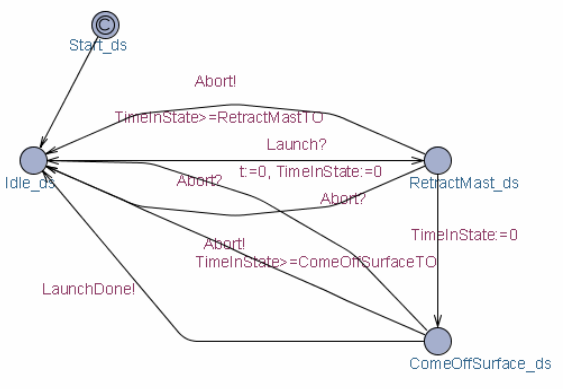


Figure 35: Launcher module in UPPAAL

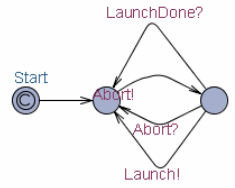


Figure 36: Driver for Launcher module in UPPAAL

E<> Launcher_P.RetractMast_ds and Launcher_P.TimeInState >= Launcher_P.RetractMastTO imply Launcher_P.Idle_ds

/*Is it possible to abort launcher operation when time at that state is greater than time to retract mast indicating some failure */

E<> Launcher_P.RetractMast_ds imply Launcher_P.ComeOffSurface_ds

/* Is it possible to dive down using the launcher */

E<> Launcher_P.ComeOffSurface_ds imply Launcher_P.Idle_ds

/*Is it possible to reach the final state once diving of the surface is done */

3.4.7 Verification of PayloadDelivery module

The value of level *i* remains 2. The next module selected is the PayloadDelivery subsystem at level 2. The environment for PayloadDelivery susbsystem is now abstracted. The PayloadDelivery subsystem is shown in Figure 37, the abstracted commanding environment is shown in Figure 38 and the abstracted commanded environment is shown in Figure 39.

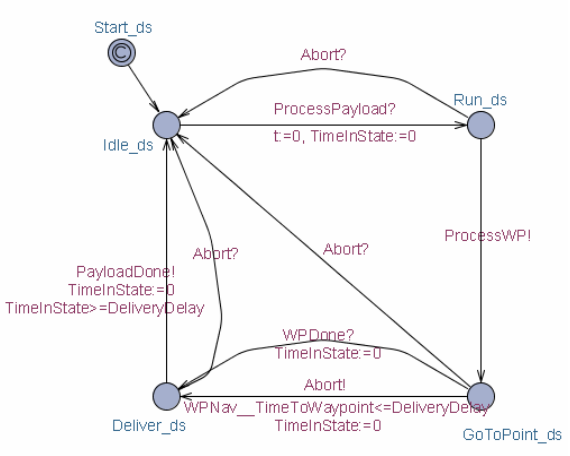


Figure 37: PayloadDelivery module in UPPAAL

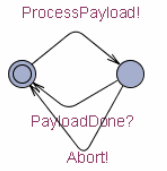


Figure 38: Driver for PayloadDelivery module in UPPAAL

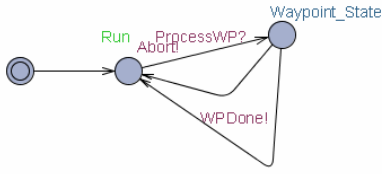


Figure 39: Stub for PayloadDelivery module in UPPAAL

Queries to verify properties satisfied by the PayloadDelivery subsystem

A[] not deadlock

/*Does there exist a deadlock*/

E<> PayloadDelivery_P.Idle_ds

/* Is it possible to reach the final state from any other state */

E<> PayloadDelivery_P.GoToPoint_ds and Stub.Waypoint_State

/*Is it that Payload module passes control to waypointnavigator to go to the desired location*/

E<> PayloadDelivery_P.Deliver_ds and Stub.Run

/*Synchronizes with waypointnavigator to go to the desired location successfully*/

E<> PayloadDelivery_P.GoToPoint_ds and WPNav__TimeToWaypoint <= PayloadDelivery_P.DeliveryDelay imply PayloadDelivery_P.Deliver_ds and Stub.Run

/*With the guard conditions being satisfied does the payloaddelivery synchronize with waypointnavigator successfully */

3.4.8 Verification of DeviceCommander module

The value of level *i* remains 2. The next module selected is the DeviceCommander subsystem at level 2. The environment for DeviceCommander susbsystem is now abstracted. The DeviceCommander subsystem is shown in Figure 40, and the abstracted commanding environment is shown in Figure 41.

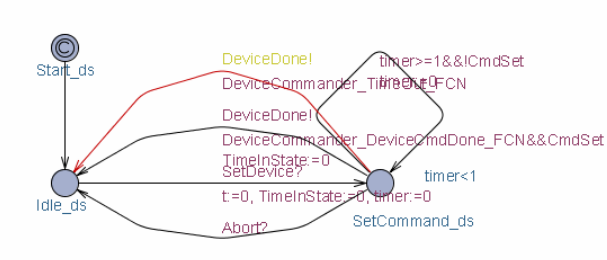


Figure 40: DeviceCommader module

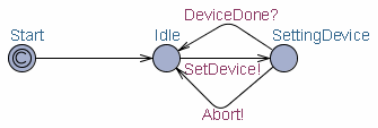


Figure 41: Driver for DeviceCommander module in UPPAAL

Queries for DeviceCommander module is as given below.

A[] not deadlock

/*Does there exist a deadlock*/

E<> DeviceCommander_P.SetCommand_ds and DeviceCommander_P.timer<1

/* s it possible for time to be greater than 1 at state setcommand*/

E<> DeviceCommander_P.DeviceCommander_DeviceCmdDone_FCN && DeviceCommander_P.CmdSet imply DeviceCommander_P.Idle_ds

/*Is device set successfully*/

E<> *DeviceCommander_P.DeviceCommander_TimeOut_FCN* *&&* *DeviceCommander_P.SetCommand_ds imply DeviceCommander_P.Idle_ds*

/* Is it possible to time out when going to set a device*/

E<> DeviceCommander_P.SetCommand_ds and DeviceCommander_P.timer>1

/* Is it possible to remain at state SetCommand without updation for more than 1 second*/

3.4.9 Verification of Pause module

The value of level *i* remains 2. The next module selected is the PayloadDelivery subsystem at level 2. The environment for PayloadDelivery susbsystem is now abstracted. The PayloadDelivery subsystem is shown in Figure 42, and the abstracted commanding environment is shown in Figure 43.

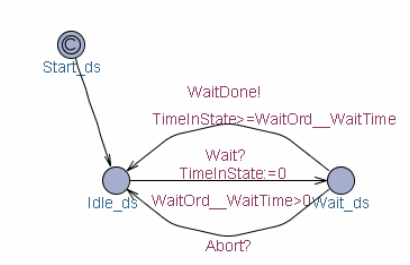


Figure 42: Pause module in UPPAAL

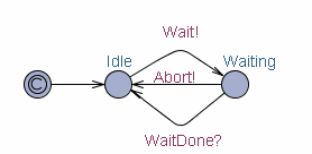


Figure 43: Driver for pause module in UPPAAL

Queries to verify properties satisfied by the Pause subsystem

A[] not deadlock

/*Does there exist a deadlock*/

Pause_P.Idle_ds and WaitOrd__WaitTime>0 --> Pause_P.Wait_ds

/* Is it possible to go to wait state to keep the coordinator waiting */

E<> Pause_P.Wait_ds and Pause_P.TimeInState>=WaitOrd__WaitTime imply Pause_P.Idle_ds

/* Is it possible to successfully complete pause operation*/

3.4.10 Verification of Sequential coordinator module

The value of level *i* changes to level 3. The next module selected is the Sequential Coordinator subsystem at level 2. The environment for sequential coordinator susbsystem is now abstracted. The Sequential Coordinator subsystem is shown in Figure 44, the abstracted commanding environment is shown in Figure 45 and the abstracted commanded environment is shown in Figure 46.

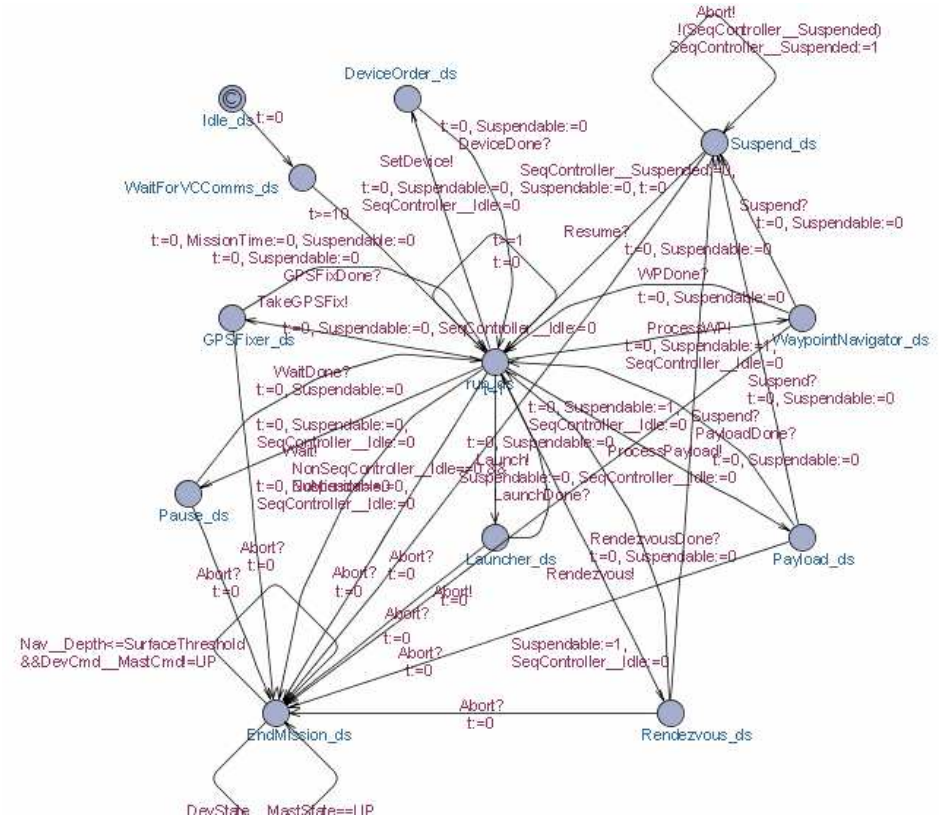


Figure 44: Sequential coordinator module in UPPAAL

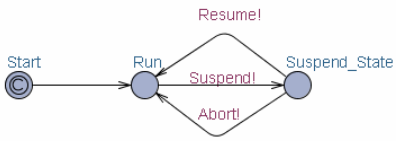


Figure 45: Driver for sequential coordinator module in UPPAAL

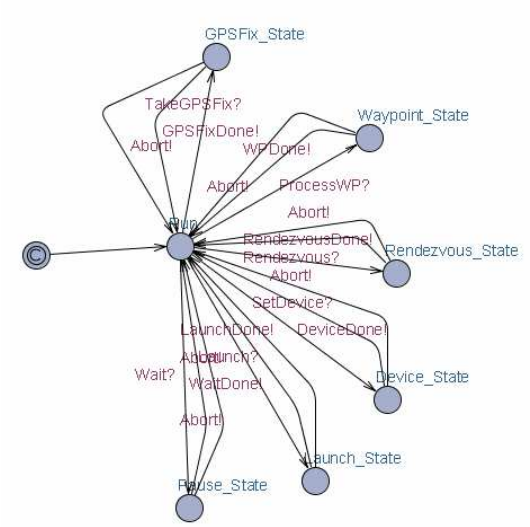


Figure 46: Stub for Sequential coordinator module in UPPAAL

Queries to verify properties satisfied by the Sequential Coordinator subsystem

A[] not deadlock

/*Does there exist a deadlock*/

E<> Controller_P.EndMission_ds

/*Is it possible to finally reach the end state*/

E<> Controller_P.WaitForVCComms_ds and Controller_P.t>10

/*Does the sequential controller wait for 10 seconds before it goes to run state*/

E<> Controller_P.run_ds and Controller_P.t<=1

/*Does the controller check the missionqueue every 1 second to check for pending requests*/

E<> Controller_P.GPSFixer_ds and Stub.GPSFix_State

/*Does the controller pass control to GPSfixer operation controller when the order is to perfom GPSFix*/

E<> Controller_P.GPSFixer_ds and Controller_P.Suspendable==0

/*Is it that GPSFixer is non suspendable(Yes indicating the design is correct as it should not be suspendable)*/

E<> Controller_P.GPSFixer_ds and SeqController__Idle==0

/*Is it possible for sequential coordinator to transfer control to waypoint navigator and wait*/

E<> Controller_P.run_ds imply Stub.Run

/*Is it possible that when controller is in run state GPSFixer is in idle state indicating that GPSFix has been done*/

E<> Controller_P.WaypointNavigator_ds and Stub.Waypoint_State

/*Does the controller pass control to WaypointNavigator operation controller when the order is to perform Waypoint navigation*/

E<> Controller_P.WaypointNavigator_ds and Stub.Run

/*Does the WaypointNavigator perform the operation successfully*/

E<> Controller_P.WaypointNavigator_ds and Controller_P.Suspendable==1

/*Is it that WaypointNavigator is suspendable(Yes indicating the design is correct as it should be suspendable)*/

E<> Controller_P.WaypointNavigator_ds and SeqController__Idle==0

/*Is it possible for sequential coordinator to transfer control to waypoint navigator and wait*/

E<> Controller_P.Pause_ds and Stub.Pause_State

/*Does the sequential controller pass control to pause*/

E<> Controller_P.Pause_ds and SeqController__Idle==0

/*Is the Secontroller idle when it passes control to Pause*/

E<> Controller_P.Pause_ds and Controller_P.Suspendable==0

/*Is the pause operation suspendable*/

E<> Controller_P.Launcher_ds and Stub.Launch_State

/*Does the controller pass control to Launcher module */

E<> Controller_P.run_ds and Stub.Run

/*Is Launch command completed successfully*/

E<> Controller_P.Launcher_ds and SeqController__Idle==0

/*Is the Seqcontroller idle when it passes control to Launcher*/

E<> Controller_P.Launcher_ds and Controller_P.Suspendable==0

/*Is the Launch operation suspendable*/

E<> Controller_P.Rendezvous_ds and Stub.Rendezvous_State

/*Is it possible to pass control to Rendezvous*/

E<> Controller_P.run_ds and Stub.Run

/*Is the Rendezvous mission completed successfully */

E<> Controller_P.Rendezvous_ds and SeqController__Idle==0

/*Is the Seqcontroller idle when it passes control to Rendezvous*/

E<> Controller_P.Rendezvous_ds and Controller_P.Suspendable==1

/*Is the Launch operation suspendable*/

E<> Controller_P.DeviceOrder_ds and Stub.Device_State

/*Is it possible to pass control to DeviceCommander module*/

E<> Controller_P.DeviceOrder_ds and SeqController__Idle==0

/*Is the Seqcontroller idle when it passes control to Devicecommander*/

E<> Controller_P.DeviceOrder_ds and Controller_P.Suspendable==0

/*Is the DeviceCommander operation suspendable*/

E<> Controller_P.Payload_ds and SeqController__Idle==0

/*Is the Seqcontroller idle when it passes control to Payload*/

E<> Controller_P.Payload_ds and Controller_P.Suspendable==0

/*Is the Payload operation suspendable*/

E<> Controller_P.Suspend_ds and SeqController__Suspended==1

/*Is there a method to test whether the seq.Controller is suspended*/

E<> Controller_P.Suspend_ds imply Controller_P.run_ds or Controller_P.EndMission_ds

/*Is it possible to return back to normal operation or end the mission after suspension*/

E<> Nav__Depth<=Controller_P.SurfaceThreshold && DevCmd__MastCmd != Controller_P.UP

/*Is it possible to come to surface of water and raise mast to indicate end of mission*/

E<> Controller_P.EndMission_ds and DevState__MastState==Controller_P.UP

/*Is it possible to raise mast at end of state*/

E<> Controller_P.run_ds && NonSeqController__Idle==0 && NoMission==0 imply Controller_P.EndMission_ds

/*Does the Seq. Controller check for the status of other controllers before ending the mission*/

E<> Controller_P.EndMission_ds and Controller_P.Suspendable==0

/*Is the end mission state suspendable*/

E<> Controller_P.EndMission_ds and SeqController__Idle==0

/*Is the Seq. Controller idle at end mission state*/

E<> Driver.Suspend_State imply Controller_P.Suspend_ds

/*Is it possible to suspend Seq. Controller by Non Seq. Controller*/

E<> Driver.Run and Controller_P.run_ds

/*Is it possible for both the Seq. and Non Seq. Controller to be ready at the same time*/

3.4.11 Verification of Timed Coordinator module

The value of level *i* remains at level 3. The next module selected is the Timed Coordinator subsystem at level 2. The environment for Timed coordinator susbsystem is now abstracted. The Timed Coordinator subsystem is shown in Figure 47, and the abstracted commanded environment is shown in Figure 48.

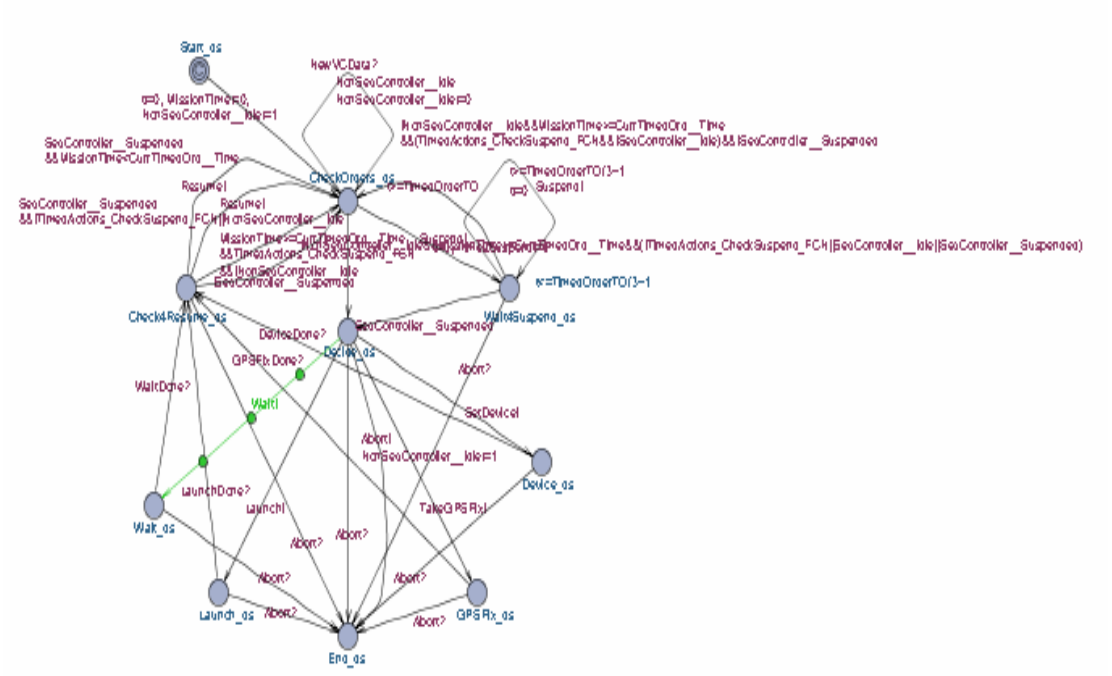


Figure 47: Timed coordinator module in UPPAAL

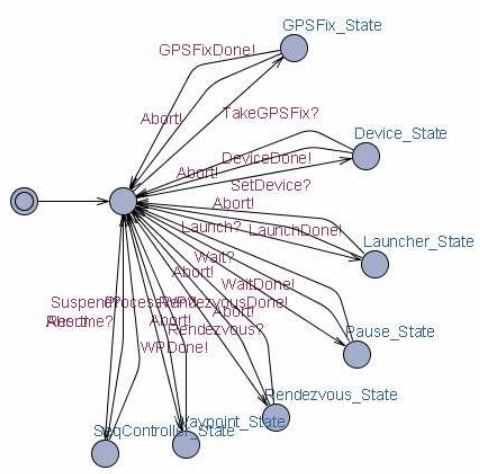


Figure 48: Stub for timed coordinator module in UPPAAL

Queries to verify the the properties satisfied by the Timed coordinator

A[] not deadlock

/*Does there exist a deadlock*/

E<> TimedActions_P.End_ds

/*Is it possible to reach the final state*/

E<> TimedActions_P.Device_ds imply Stub.Device_State

/*Does the Timed action pass the control to Device commander when it needs to set or start a device*/

E<> TimedActions_P.Launch_ds and Stub.Launcher_State

/*Does the Timed action pass the control to launcher when it needs to comeoffsurface or act with the mast*/

E<> TimedActions_P.Wait_ds imply Stub.Pause_State

/*Does the Timed action pass the control to pause when it needs to wait*/

E<> TimedActions_P.GPSFix_ds imply Stub.GPSFix_State

/*Is it possible to execute a timed GPSFix*/

E<> TimedActions_P.Wait4Suspend_ds and TimedActions_P.t == TimedActions_P.TimedOrderTO/3+1

/*Does the timed actions try to suspend the seq. coordinator every desired time*/

E<> TimedActions_P.Wait4Suspend_ds imply TimedActions_P.t <= TimedActions_P.TimedOrderTO/3+1

/*Does the timed actions try to suspend the seq. coordinator every desired time*/

E<> TimedActions_P.CheckOrders_ds && TimedActions_P.MissionTime >= CurrTimedOrd__Time&&(TimedActions_P.InterruptCoordinator_CheckSuspend_FCN&&SeqCoordinator__Suspendable&&!SeqCoordinator__Idle)&& !SeqCoordinator__Suspended imply TimedActions_P.Wait4Suspend_ds

/*Is it possible to suspend the seq. controller*/

E<> TimedActions_P.MissionTime>=CurrTimedOrd__Time && TimedActions_P.TimedActions_CheckSuspend_FCN && !NonSeqController__Idle imply TimedActions_P.CheckOrders_ds

/*Does the Timed action check for orders when the mission time is greater than the current time (indicating timed mission

can be accomplished) and does Timed action check the need to suspend seq controller or not and timed actions is not idle*/

E<> TimedActions_P.Check4Resume_ds and !SeqController__Suspended imply TimedActions_P.CheckOrders_ds

/*Is it possible for the timed action to chekc for orders when the sequential controller doesnt need to be suspended*/

E<> TimedActions_P.Check4Resume_ds and SeqController__Suspended and TimedActions_P.MissionTime<CurrTimedOrd__Time imply TimedActions_P.CheckOrders_ds

/*Does the Timed action controller check timed orders when Seq. Controller is suspended and Mission time is greater than
current time for timed order*/

E<> TimedActions_P.Check4Resume_ds and SeqController__Suspended && not TimedActions_P.TimedActions_CheckSuspend_FCN || NonSeqController__Idle imply TimedActions_P.CheckOrders_ds

/*Does the Time action controller check timed orders when seq. controller is suspended and timed action doesn't need suspension or timed controller is idle*/

E<>TimedActions_P.CheckOrders_ds and !NonSeqController__Idle && TimedActions_P.MissionTime>=CurrTimedOrd__Time && (not TimedActions_P.TimedActions_CheckSuspend_FCN || SeqController__Idle || SeqController__Suspended) imply TimedActions_P.Decide_ds

/*Does the timed action become ready to execute orders when the timed controller is not idle and mission time is greater than current timed order time and timed action doesnt need suspension or sequential controller is idle or sequential controller is already suspended*/

E<> TimedActions_P.Wait_ds imply not Stub.Pause_State

/*Is it possible to transfer control to Pause moduel*/

E<> TimedActions_P.Decide_ds imply TimedActions_P.End_ds

/*Is it possible to go to the final state from the decide state*/

E<> TimedActions_P.Wait_ds imply TimedActions_P.End_ds

/*Is it possible to go to the final state from the situation where control is passed to the pause controller*/

E<> TimedActions_P.Launch_ds imply TimedActions_P.End_ds

/*Is it possible to go to the final state from the situation where control is passed to the launch controller*/

E<> TimedActions_P.GPSFix_ds imply TimedActions_P.End_ds

/*Is it possible to go to the final state from the situation where control is passed to the GPSFix controller*/

E<> TimedActions_P.Device_ds imply TimedActions_P.End_ds

/*Is it possible to go to the final state from the situation where control is passed to the Device controller*/

E<> TimedActions_P.CheckOrders_ds imply TimedActions_P.End_ds and TimedActions_P.Idle==1

/*Is it possible to go to the final state from state to check orders*/

A<> TimedActions_P.Decide_ds imply (TimedActions_P.End_ds || TimedActions_P.Wait_ds || TimedActions_P.Launch_ds || TimedActions_P.GPSFix_ds || TimedActions_P.Device_ds)

/*All paths eventually lead to final state or pause or launcher or GPSFixer or Device from the decide state in timed actions*/

E<> TimedActions_P.End_ds imply TimedActions_P.Idle==1

/*Is it possible for Non Seq. Controller to be idle at end of mission*/

E<> TimedActions_P.CheckOrders_ds imply TimedActions_P.Idle==0 && TimedActions_P.Done

/*Is it possible for Non Seq.Controller to be idle at CO state*/

E<> TimedActions_P.CheckOrders_ds imply TimedActions_P.MissionTime==0

/*Is the mission time zero at CO state*/

3.4.12 Verification of Safety Coordinator module

The value of level *i* remains at level 3. The next module selected is the Safety Coordinator subsystem at level 2. The Safety Coordinator checks the voltage, depth of water and the functioning of other devices from the common database. The Safety Coordinator is as shown in Figure 49.

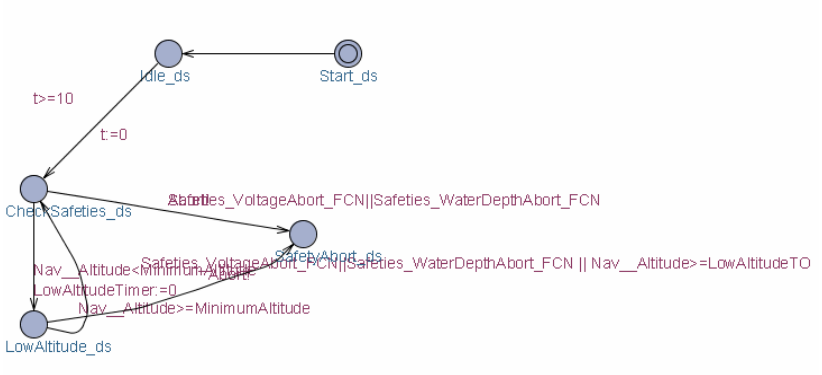


Figure 49: Safety coordinator module in UPPAAL

Queries to verify the properties of the safety module

A[] not deadlock

/*Does there exist a deadlock*/

E<> Safeties_P.CheckSafeties_ds && Nav__Altitude < Safeties_P.MinimumAltitude imply Safeties_P.LowAltitude_ds

/*Is there method a to check water depth safety*/

E<> Safeties_P.CheckSafeties_ds && Safeties_P.Safeties_VoltageAbort_FCN || Safeties_P.Safeties_WaterDepthAbort_FCN) imply Safeties_P.SafetyAbort_ds

/*Is tehr method to check voltage safety*/

E<> Safeties_P.LowAltitude_ds && Nav__Altitude >= Safeties_P.MinimumAltitude imply Safeties_P.CheckSafeties_ds

/*Is it possible to correct the altitude*/

E<> Safeties_P.LowAltitude_ds && Safeties_P.Safeties_VoltageAbort_FCN ||Safeties_P.Safeties_WaterDepthAbort_FCN || Nav__Altitude >= Safeties_P.LowAltitudeTO imply Safeties_P.SafetyAbort_ds

/*Is there a method to abort mission if safety is violated*/

4 Model-based Animation/Simulation

Animation/Simulation is the imitation of the reality for studying the effect of changing parameters in a model as a means of taking a decision. Animation/Simulation imitates or estimates how events might occur in a real situation. It can involve complex mathematical modeling, role playing without the aid of technology, or combinations. The value lies in placing one under realistic condition, that change as a result of behavior of other variables involved so one cannot anticipate the sequence of events or the final outcome. An animation/simulation tool aids in the creation of the realistic environment using mathematical formulas and algorithms. Usage of a simulation tool gives one the advantage of checking the accuracy of algorithms involved in the experiment without incurring damage to the vehicle or equipments involved. Due to which we looked into the feasibility of creating an animation/simulation tool for a mission driven AUV which we discuss in this section.

A simulation tool had been implemented for the automated highway system in the PATH project at Berkley [67] [68]. The use of such a simulation tool proved to be advantageous because of the beneficial outcomes. It helped to prove the correctness of the mathematical modeling before actual implementation. A simulation in addition to verification further strengthens the correctness of a modeled system. This is because mostly verification of a hybrid system is carried out by abstracting the system and it might miss out on faulty situation which might occur for the combined system. Animation/Simulation involves both the continuous and discrete dynamics combined together to successfully execute a mission. So a simulation would catch some interactions which might be missed while carrying out verification as described by Godbole, Lygeros and Sastry in [69].

We looked into the feasibility of creating a simulation environment to model the actions executed by the AUV under given mission orders. The preliminary tool developed is a very basic tool which proves the possibility of having a very advanced simulation tool in future. The simulation tool is very specific to the survey missions for an AUV. OpenGL is used to simulate/animate the missions executed by the AUV.

4.1 OpenGL: Tool for Animation/Simulation

OpenGL is a software interface to the graphics hardware (GL stands for Graphics library). OpenGL is a hardware independent interface to be implemented on many

different hardware platforms. OpenGL contains commands to draw geometric primitives like points, lines, and polygons to build the desired model. The desired model can be any complicated shape like that of an automobile, aeroplane etc. It allows the creation of interactive programs with colored images of moving three-dimensional objects. It can be used to control advanced graphics technology to create realistic images. OpenGL provides a set of commands that allow the specification of geometric objects in two or three dimensions, using the provided primitives, together with commands that control how these objects are rendered into the frame buffer and also to create interactive applications with these commands.

OpenGL is like a state machine, the state being defined by color, current viewing, projection transformation, polygon drawing mode, characteristics of light etc. The state remains the same until changed by changing the parameters within the functions used to design the desired model. OpenGL also supports animation of graphical models drawn. Thus using OpenGL we can move or rotate or involve translation of an object the way we want.

GLUT (Open**GL** **U**tility **T**oolkit) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system and provides many modeling features. Functions performed include window definition, window control, and monitoring of keyboard and mouse input. Routines for drawing a number of geometric primitives (both in solid and wireframe mode) are also provided, including cubes, spheres. GLUT even has some limited support for creating pop-up windows.

A typical program that uses OpenGL begins with calls to open a window into the frame buffer into which the program will draw. Then, calls are made to allocate a GL context and associate it with the window. Once a GL context is allocated, OpenGL commands can be issued. Some calls are used to draw simple geometric objects (i.e. points, line segments, and polygons), while others affect the rendering of these primitives including how they are lit or colored and how they are mapped from the user's two- or three-dimensional model space to the two-dimensional screen.

The major graphics operations which OpenGL performs to render (render is the process by which a computer creates images from models) an image on the screen are as given next. Construct shapes from geometric primitives, thereby creating mathematical

descriptions of objects. (OpenGL considers points, lines, polygons, images, and bitmaps to be primitives.). Arrange the objects in three-dimensional space and select the desired vantage point for viewing the composed scene. Calculate the color of all the objects. The color might be explicitly assigned by the application, determined from specified lighting conditions, obtained by pasting a texture onto the objects, or some combination of these three actions. Convert the mathematical description of objects and their associated color information to pixels on the screen. This process is called *rasterization*. During these stages, OpenGL might perform other operations, such as eliminating parts of objects that are hidden by other objects. In addition, after the scene is rasterized but before it's drawn on the screen, one can perform some operations on the pixel data.

The final rendered image consists of pixels drawn on the screen. A pixel is the smallest visible element the display hardware can put on the screen. Information about the pixels (for instance, what color they're supposed to be) is organized in memory into bitplanes. A bitplane is an area of memory that holds one bit of information for every pixel on the screen. For example the bit might indicate how red a particular pixel is supposed to be. The bitplanes are themselves organized into a *framebuffer*, which holds all the information that the graphics display needs to control the color and intensity of all the pixels on the screen.

Most implementations of OpenGL have a similar order of operations, a series of processing stages called the OpenGL rendering pipeline.

Display lists all data, whether it describes geometry or pixels, can be saved in a *display list* for current or later use. (The alternative to retaining data in a display list is processing the data immediately - also known as *immediate mode*.) When a display list is xecuted, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions. Evaluators provide a method to derive the vertices used to represent the surface from the control points. The method is a polynomial mapping, which can produce surface normal, texture coordinates, colors, and spatial coordinate values from the control points.

For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some vertex data (for example, spatial coordinates) are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen. If advanced features are enabled, this stage is even busier. If texturing is used, texture coordinates may be generated and transformed here. If lighting is enabled, the lighting calculations are performed using the transformed vertex, surface normal, light source position, material properties, and other lighting information to produce a color value.

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half-space, defined by a plane. Point clipping simply passes or rejects vertices; line or polygon clipping can add additional vertices depending upon how the line or polygon is clipped. In some cases, this is followed by perspective division, which makes distant geometric objects appear smaller than closer objects. Then viewport and depth (z coordinate) operations are applied. If culling is enabled and the primitive is a polygon, it then may be rejected by a culling test. Depending upon the polygon mode, a polygon may be drawn as points or lines. The results of this stage are complete geometric primitives, which are the transformed and clipped vertices with related color, depth, and sometimes texture-coordinate values and guidelines for the rasterization step.

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture emory or sent to the rasterization step. If pixel data is read from the frame buffer, pixel-transfer operations (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory. There are special pixel copy operations to copy data in the framebuffer to other parts of the framebuffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture memory or back to the framebuffer.

An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. If several texture images are used, it's wise to put them into texture objects so that you can easily switch among them. Some OpenGL

implementations may have special resources to accelerate texture performance. There may be specialized, high-performance texture memory. If this memory is available, the texture objects may be prioritized to control the use of this limited and valuable resource.

Rasterization is the conversion of both geometric and pixel data into *fragments*. Each fragment square corresponds to a pixel in the framebuffer. Line and polygon stipples, line width, point size, shading model, and coverage calculations to support antialiasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square. Before values are actually stored into the framebuffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled.

The first operation which may be encountered is texturing, where a texel (texture element) is generated from texture memory for each fragment and applied to the fragment. Then fog calculations may be applied, followed by the scissor test, the alpha test, the stencil test, and the depth-buffer test (the depth buffer is for hidden-surface removal). Failing an enabled test may end the continued processing of a fragment's square. Then, blending, dithering, logical operation, and masking by a bitmask may be performed. Finally, the thoroughly processedfragment is drawn into the appropriate buffer, where it has finally advanced to be a pixel and achieved its final resting place.

4.2 Proposed Approach for Animation/Simulation

We created a converter written in Perl. The converter takes the coordinator modules involved in a specific mission as its input. The coordinator modules are modeled using real time verification tool Uppaal which are .xml files. After opening the coordinator modules the converter extracts important information from the UPPAAL files and generates a graphics file in OpenGL. The information extracted are the different events received or sent, and the variables used.

The converter searches the sequential/timed coordinator file, extracts the mission name which is an event on a transition within the sequential/timed coordinator. Then the converter searches the file among the set of input files which models a transition on the same event. The converter then keeps extracting and expanding the sequence of events.

The expansions model the sequence of actions (algorithms) executed by the concerned controller modules.

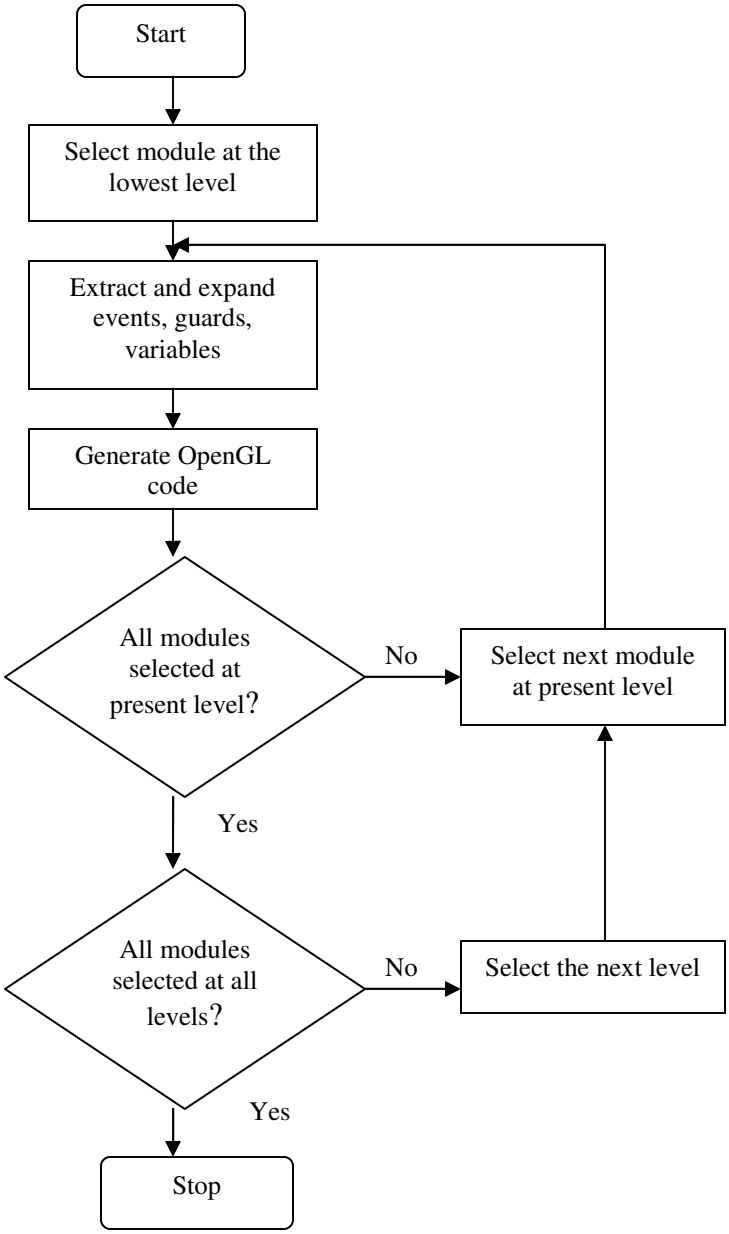
The code that is generated can then be run using the commands used to run an OpenGL program. The parameters required for a mission can be changed within the files which changes certain actions executed by the AUV. A brief description of OpenGL, converter code and explanation for one mission is explained next. The OpenGL modules are attached in appendix C.

We use a bottom up approach for simulation. We first simulate the actions implemented by the lower level controllers. Once the lower level controllers are simulated we combine the higher level controllers with the lower ones. We used this approach because the parts of mission executed by the lower level controllers are called for by the higher level ones. So this gives us an organized way to build up the simulation of the model.

The OpenGL modules contain codes which expand each of the sequences to be executed to successfully complete a mission. In the present simulation model sensor values are stored in common files. The modules collect the sensor information and other parameter changes from within the common files accessed. The modules then execute the sequence of actions according to the inputs received. After completing the mission the changed parameters are then written back to the common files like time, position etc. this way the information gets updated. The next operation or mission to be executed gathers information from the common files before starting to execute the sequence of actions. The algorithm used for the conversion of the Uppaal modules to the OpenGL code is as given below. Then follows a flowchart which shows the way the converter works and then follows a few screen shots like the AUV with the mast underwater (Figure 50) raising mast after reaching surface of water (Figure 51) and the OpenGL code for steering module.

- For i = n to 1 (where n is the lowest level)
    - For k = 1 to m (where m is the total number of modules within a level)
        - Input the hybrid model $\boldsymbol{H}$ at the $Level_i$ to the converter
        - The converter extract events σ, guard g_e and variables *Vars*
        - Generate OpenGL code to model the events, guards using the variables involved.

- Next k
- Next i



.

Flowchart 1: Method involved in simulation

Figure 50: AUV (green) with mast (yellow) underwater

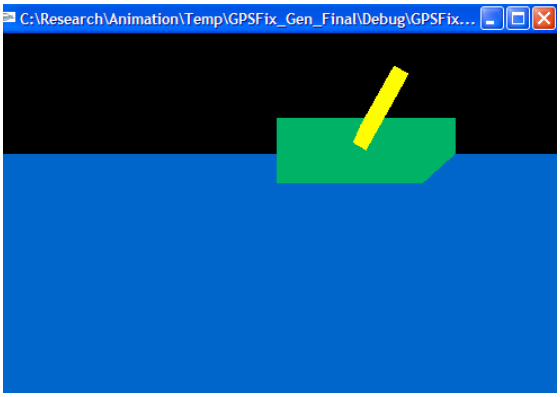


Figure 51: AUV raising mast at surface of water while executing GPSFix

The code for the Steer module follows next.

4.3 The Converter code for Steering module

#Initialization

```
#!/usr/bin/perl -w
$in = '';
$input = '';
$i = 0;
$SteeringFunctionDone = 0;
$getvariable = 0;#Variable used to get the number of variables declared in the controller modules
$first = 0; #Variable to constrain the number of times the initialization code is generated
$Iterationnumber = 0;
$Iterationnumber1 = 0;
$numbertimesdecl = 0;
$abortNumber = 0;
$numberint = 0;
$stringofdecl = '';
@arrayofdecl = ();
$stringid = '';
```

```
%sepvalue =();
@actualvar = ();
@stringofvar = ();
$SteeringSteerNumber = 0; #Variable to note down the number of times event Steer occurs
$InSteer = '';
open(OUTFILE, ">Steering.c");
open(SF, "Steering.xml") or die "Can't find file. Name it as Steering.xml\n";
open(STEERFILE, "Steering.xml");

# Finding the number of times the event Steer occurs
while ($InSteer = <SF>)
{
      if ($InSteer =~ />Steer\W</ ) # Checking for the event Steer
      {
            $SteeringSteerNumber++;
                  print "Number of time Steering $SteeringSteerNumber \n";
      }

}

# Input the Steering controller module and generate the initalization code
while($in = <>) # Opening the Steer controller module
{
          if($first == 0) # Checking whether this code is generated once or not
          {
                  print OUTFILE "\n/********* Steering module sequence animation**************/";
                  print OUTFILE "\n#include <GL/glut.h>";
           print OUTFILE "\n#include <stdlib.h>";
           print OUTFILE "\n#include<stdio.h>";
                  print OUTFILE "\n#include<time.h>";
                  print OUTFILE "\n#include<math.h>";

           print OUTFILE "\n#define VMR 0.001 //Rate to move up depeding";
           print OUTFILE "\n#define HMR 0.001 //Rate to move up depeding";
           print OUTFILE "\n#define SurfaceThreshold 1 //To assign the boundary for water surface";
           print OUTFILE "\n#define MaxAngle 90 //To assign the mast the maximum angle";
                  print OUTFILE "\n#define MastRate 0.05// Rate at which mast is raised";
                  print OUTFILE "\n#define UP 1// To check whether mast is raised or not";
                  $first++;
          }

# Check for the number of declaration of variables
          while($input = <STEERFILE>)
                  {
                  if ($input =~ /\sint\s/)
                  {
                        $numbertimesint++; # Keeps track of iterations
                              print"Integers = $numbertimesint\n";
                  }
                  }
# Check for patterns as within declaration
          if($in =~ /int\s\S+:=\d/)
      {
          $stringofdecl = $&; # Store the pattern in a string
                  @arrayofdecl = split(' ', $stringofdecl); # Get rid of int
```

```
        $stringofvar = $arrayofdecl[1]; # Get the element of the array which contains the string of variables
        @actualvar = split(',',$stringofvar); # split it based on commas to get the total no of variables

         for ($count = 0; $count<=$#actualvar; $count++) # Iterate for all the variables
         {
              $stringid = $actualvar[$count]; # Store each variable in a string
              %sepvalue = split(':',$stringid); # Split variables based on colon
              print OUTFILE "static GLfloat "; print OUTFILE %sepvalue; print OUTFILE ";"; print OUTFILE "\n";

         }
         $getvariable++;
        }
```

 # Generate all the variables extracted and the initialization modules

```
        if ($Iterationnumber == 0 && $getvariable == $numbertimesint)
        {
                print OUTFILE "\nstatic GLfloat MastAngle;";
                print OUTFILE "\nstatic GLfloat MastMotion = 0.0;";
                print OUTFILE "\nstatic GLfloat Slope; // To get the slope of the line in direct steeringmode";
                print OUTFILE "\nstatic GLfloat LatLongDiff; /*The constant in the equation of a line for direct steermode*/";
                print OUTFILE "\nstatic GLfloat timer = 0;";

                print OUTFILE "\nfloat TimeOfOperation, TotalTime;";
                print OUTFILE "\nint SteerMode;";
                print OUTFILE "\nint z=0;";
                print OUTFILE "\nint Steering, count;";
                print OUTFILE "\nint i = 0 ; //Used for abort signal";
                print OUTFILE "\nint j =0, k =0, n =0, l =1;//j for lat loop, k for long loop, l for LINE loop";
                print OUTFILE "\nint Lat = 0, begin, o = 0, m = 0, DevState__MastState;";
                print OUTFILE "\nint MastUp = 0;";
                print OUTFILE "\nfloat temp1, temp2, square;";
         print OUTFILE "\nfloat temp3, number, ExitTime,squareroot;";
                print OUTFILE "\ntime_t start, start1, start2, start3, start4;";
                print OUTFILE "\ntime_t end, end1, end2, end3, end4;";
                print OUTFILE "\ndouble elapsed;";
                print OUTFILE "\nFILE *STOutput, *SOutput, *SteerOutput, *StAngle;";
                print OUTFILE "\nvoid init(void)";
                print OUTFILE "\n{";
                print OUTFILE "\n        glClearColor (0.0, 0.0, 0.0, 0.0);";
                print OUTFILE "\n        glShadeModel (GL_FLAT);";
                print OUTFILE "\n}";
                print OUTFILE "\nvoid display(void)";
                print OUTFILE "\n{";
                print OUTFILE "\n        glClear (GL_COLOR_BUFFER_BIT);";
                print OUTFILE "\n        glColor3f(0.0, 0.4, 0.8);";
                print OUTFILE "\n        glRectf(-8.0, -4.0, 8.0, 2.0);";
                print OUTFILE "\n        glPushMatrix();";
                print OUTFILE "\n        glTranslatef (-1.0, 0.0, 0.0);";
                print OUTFILE "\n        glTranslatef (1.0, 0.0, 0.0);";
                print OUTFILE "\n        glTranslatef(FromLongitude, FromLatitude, 0.0);";
                print OUTFILE "\n        glTranslatef(FromLongitude, FromLatitude, 0.0);";
```

```
                print OUTFILE "\n        glPushMatrix();";
                print OUTFILE "\n        glScalef (2.0, 0.4, 1.0);";
                print OUTFILE "\n        glColor3f(0.0, 0.7, 0.4);";
                print OUTFILE "\n        glutSolidCube (1.0);";
                print OUTFILE "\n        glPopMatrix();";
                print OUTFILE "\n        glTranslatef (0.0, 0.1, 0.0);";
                print OUTFILE "\n        glTranslatef(0.08*FromLongitude, MastMotion, 0.0);";
                print OUTFILE "\n        glRotatef(0.0, 0.0, 0.0, 1.0);";
                print OUTFILE "\n        glTranslatef(0.5, MastMotion, 0.0);";
                print OUTFILE "\n        glPushMatrix();";
                print OUTFILE "\n        glScalef (0.8, 0.2, 0.5);";
                print OUTFILE "\n        glColor3f(1.0, 1.0, 0.0);";
                print OUTFILE "\n        glutSolidCube(1.0);";
                print OUTFILE "\n        glPopMatrix();";
                print OUTFILE "\n        glPopMatrix();";
                print OUTFILE "\n        glutSwapBuffers();";
                print OUTFILE "\n}";
                $Iterationnumber++;

        }
# Extract all the information for the Steer mission
        if($in =~ />Steer\W</)
        {
                print OUTFILE "\n/*Module to Steer the AUV*/";
                print OUTFILE "\nvoid Steer(void)";
                print OUTFILE "\n{";
                print OUTFILE "\n        if(begin == 0 )";
                print OUTFILE "\n        {";
                print OUTFILE "\n                time(&start);";
                print OUTFILE "\n                time(&start1);";
                print OUTFILE "\n                time(&start2);";
                print OUTFILE "\n                time(&start3);";
                print OUTFILE "\n                time(&start4);";
                print OUTFILE "\n                begin++;";
                print OUTFILE "\n        }";
                print OUTFILE "\n        if (Steering == 1)";
                print OUTFILE "\n        {";
                print OUTFILE "\n         /*AUV moving horizontally when FromLongitude is lesser
than ToLongitude*/";
                print OUTFILE "\n                if ((FromLongitude < ToLongitude) && (k == 0))";
                print OUTFILE "\n                {";
                 print OUTFILE "\n        /*AUV for LINE steering mode to get slope*/";
                print OUTFILE "\n                        if ((SteerMode == 1) && (n == 0))";
                print OUTFILE "\n                        {";
                print OUTFILE "\n                                Slope = (ToLatitude-
FromLatitude)/(ToLongitude-
FromLongitude);";
                print OUTFILE "\n                                LatLongDiff = FromLatitude -
Slope*FromLongitude;";
                print OUTFILE "\n                                n++; l--;";
                print OUTFILE "\n                        }";
                print OUTFILE "\n        /*AUV for LINE steering mode */";
                print OUTFILE "\n                        if (SteerMode == 1 && l ==0)";
                print OUTFILE "\n                        {";
                print OUTFILE "\n                                FromLongitude = FromLongitude
+ HMR;";
```

```
                print OUTFILE "\n                                        FromLatitude                    =
Slope*FromLongitude + LatLongDiff;";
                print OUTFILE "\n                                        MastMotion                      =
0.008*FromLongitude;";
                print OUTFILE "\n                                        j = 1;";
                print OUTFILE "\n                                        if(difftime(end,start1) == 2)";
                print OUTFILE "\n                                        {";
                print OUTFILE "\n";
                print OUTFILE '                                         printf("Time = %f, Steermode
LINE %f,
                                                                        %f\n",difftime(end,start1),
FromLatitude, FromLongitude);';
                print OUTFILE "\n                                               time(&start1);";
                print OUTFILE "\n                                       }";
                print OUTFILE "\n               if(ToLongitude <= FromLongitude)";
                print OUTFILE "\n               {";
                print OUTFILE "\n                       glutIdleFunc(NULL);";
                print OUTFILE "\n                       k =1;";
                print OUTFILE "\n                       l = 1;";
                print OUTFILE "\n";
                print OUTFILE '                 printf("Time = %f Steermode LINE %f,
                %f\n",difftime(end,start), FromLatitude, FromLongitude);';
                print OUTFILE "\n               }";
                print OUTFILE "\n               glutPostRedisplay();";
                print OUTFILE "\n       }";
                print OUTFILE "\n}";
                print OUTFILE "\n/*AUV moving horizontally when FromLongitude is greater than
ToLongitude*/";
                print OUTFILE "\n       if ((ToLongitude < FromLongitude ) && (k == 0) )";
                print OUTFILE "\n       {";
                print OUTFILE "\n       /*AUV for LINE steering mode to get slope*/";

                print OUTFILE "\n               if (SteerMode == 1 && n == 0)";
                print OUTFILE "\n               {";
                print OUTFILE "\n                       Slope           =       (ToLatitude-
FromLatitude)/(ToLongitude-FromLongitude);";
                print OUTFILE "\n                       LatLongDiff     =       FromLatitude    -
Slope*FromLongitude;";
                print OUTFILE "\n                       n++; l--;";
                print OUTFILE "\n               }";
                print OUTFILE "\n       /*AUV for LINE steering mode*/ ";
                print OUTFILE "\n               if (SteerMode == 1 && l == 0 )";
                print OUTFILE "\n               {";
                print OUTFILE "\n                       FromLongitude = FromLongitude - HMR;";
                print OUTFILE "\n                       FromLatitude = Slope*FromLongitude +
LatLongDiff;";
                print OUTFILE "\n                       MastMotion = 0.008*FromLongitude;";
                print OUTFILE "\n                       j = 1;";
                print OUTFILE "\n                       if((difftime(end,start2)) == 2)";
                print OUTFILE "\n                       {";
                print OUTFILE "\n";
                print OUTFILE '                         printf("Time = %f, Steermode LINE Lat =
%f,
                                                                Long = %f\n",difftime(end,start2),
FromLatitude,
                                                                FromLongitude);';
```

```
print OUTFILE "\n                                time(&start2);";
print OUTFILE "\n                        }";
print OUTFILE "\n                        if(ToLongitude >= FromLongitude )";
print OUTFILE "\n                        {";
print OUTFILE "\n                                glutIdleFunc(NULL);";
print OUTFILE "\n                                k =1;";
print OUTFILE "\n                                l = 1;";
print OUTFILE "\n";
print OUTFILE '                                 printf("Time = %f, Steermode LINE %f,
                                                %f\n",difftime(end,start),      FromLatitude,
FromLongitude);';
print OUTFILE "\n                               }";
print OUTFILE "\n                               glutPostRedisplay();";
print OUTFILE "\n                       }";
print OUTFILE "\n}";
print OUTFILE "\n       /*AUV moving Vertically when FromLatitude is lesser than
ToLatitude */";

print OUTFILE "\n       if (FromLatitude < ToLatitude && j == 0 ) ";
print OUTFILE "\n       {";
print OUTFILE "\n               FromLatitude = FromLatitude + VMR;";
print OUTFILE "\n               MastMotion = 0.008*FromLatitude;";
print OUTFILE "\n               if(difftime(end,start3) == 2)";
print OUTFILE "\n               {";
print OUTFILE "\n                       if (SteerMode == 1)";
print OUTFILE "\n";
print OUTFILE '                                 printf("Time = %f Steermode LINE
Lat:%f
                                                Long = %f\n", difftime(end,start3),
FromLatitude, FromLongitude);';
print OUTFILE "\n                       time(&start3);";
print OUTFILE "\n               }";

print OUTFILE "\n               if((FromLatitude        >=      ToLatitude)     &&
(FromLongitude == ToLongitude))";
print OUTFILE "\n               {";
print OUTFILE "\n                       glutIdleFunc(NULL);";
print OUTFILE "\n                       j = 1; ";
print OUTFILE "\n                       k = 1;";
print OUTFILE "\n";
print OUTFILE   '                       printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);';
print OUTFILE "\n               }";
print OUTFILE "\n               glutPostRedisplay();";
print OUTFILE "\n       }";
print OUTFILE "\n       /*AUV moving vertically when FromLatitude is greater than
ToLatitude */";
print OUTFILE "\n       if (FromLatitude > ToLatitude && j == 0 ) ";
print OUTFILE "\n       {";
print OUTFILE "\n               FromLatitude = FromLatitude - VMR;";
print OUTFILE "\n               MastMotion = 0.008*FromLatitude;";
print OUTFILE "\n               if(difftime(end,start4)== 2)";
print OUTFILE "\n               {";
print OUTFILE "\n                       if (SteerMode == 1)";
print OUTFILE "\n";
print OUTFILE '                 printf("Time = %f Steermode LINE Lat:%f Long =
```

```
                                        %f\n",difftime(end,start4),          FromLatitude,
FromLongitude);';
                print OUTFILE "         time(&start4);";
                print OUTFILE "\n               }";
                print OUTFILE "\n               if((FromLatitude <= ToLatitude) &&
(FromLongitude ==ToLongitude))";
                print OUTFILE "\n               {";
                print OUTFILE "\n                       glutIdleFunc(NULL);";
                print OUTFILE "\n                       j = 1; ";
                print OUTFILE "\n                       k = 1;";
                print OUTFILE "\n";

                print OUTFILE '                 printf("Time = %f, Steermode LINE %f, %f\n" ,
                                        difftime(end,start),          FromLatitude,
FromLongitude);';
                print OUTFILE "\n               }";

                print OUTFILE "\n               glutPostRedisplay();";
                print OUTFILE "\n       }";
                print OUTFILE "\n}";
                print OUTFILE "\n time(&end);";
                print OUTFILE "\n if(k == 1 && j == 1)";
                print OUTFILE "\n{";
                print OUTFILE "\n       if(count == 0)";
                print OUTFILE "\n       {";
                print OUTFILE "\n";

                print OUTFILE '                 SOutput = fopen( "C:\\Research\\ Animation \\ Temp
\\ Mission \\
                                        Position.txt", "w" );';
                print OUTFILE "\n";
                print OUTFILE '                 fprintf( SOutput, "%f\n", FromLatitude );';
                print OUTFILE "\n";
                print OUTFILE '                 fprintf( SOutput, "%f\n", FromLongitude );';
                print OUTFILE "\n";
                print OUTFILE "\n               fclose(SOutput);";

                print OUTFILE "\n";
                print OUTFILE '                 SteerOutput          =          fopen(
"C:\\Research\\Animation\\Temp\\Mission\\SteerOutput.txt
        ", "w" );';
                print OUTFILE "\n";
                print OUTFILE '                 fprintf( SteerOutput, "%f\n", FromLatitude );';
                print OUTFILE "\n";
                print OUTFILE '                 fprintf( SteerOutput, "%f\n", FromLongitude );';
                print OUTFILE "\n";
                print OUTFILE '                 fprintf(SteerOutput, "%f\n", difftime(end, start));';
                print OUTFILE "\n";
                print OUTFILE "\n               fclose(SteerOutput);";

                print OUTFILE "\n";
                print OUTFILE '                 STOutput                                        =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "r");';
                print OUTFILE "\n";
                print OUTFILE '                 fscanf(STOutput,"%f\n", &TotalTime);';
                print OUTFILE "\n               fclose(STOutput);";
```

```
                print OUTFILE "\n                    TotalTime = TotalTime + difftime(end, start);";

                print OUTFILE "\n";
                print OUTFILE '                    STOutput                                                    =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "w");';
                print OUTFILE "\n";
                print OUTFILE '                    fprintf(STOutput, "%f\n", TotalTime);';
                print OUTFILE "\n                  fclose(STOutput);";
                print OUTFILE "\n                  count = 1;";
                print OUTFILE "\n                  exit(0);";
                print OUTFILE "\n          }";
                print OUTFILE "\n}";
                print OUTFILE "\n}";


                $SteeringFunctionDone++;
        }

# Sequence of action implemented during abort
         if($in =~ />Abort\W</ && $abortNumber == 0)
        {
                print OUTFILE "\nvoid Abort(void)";
                print OUTFILE "\n{";
                print OUTFILE "\n        /*AUV moving up to take GPSFix (Control within GPSFix
module)*/";
                print OUTFILE "\n        /*Take value of SurfaceThreshold from Uppaal file */";
                print OUTFILE "\n        if (FromLatitude < SurfaceThreshold && o ==0)";
                print OUTFILE "\n        {";
                print OUTFILE "\n                FromLatitude = FromLatitude + VMR;";
                print OUTFILE "\n                MastMotion = 0.008*FromLatitude;";

                print OUTFILE "\n                if(FromLatitude == SurfaceThreshold)";
                print OUTFILE "\n                {";
                print OUTFILE "\n                        glutIdleFunc(NULL);";
                print OUTFILE "\n                        o = 1;";
                print OUTFILE "\n                }";
                print OUTFILE "\n                glutPostRedisplay();";
                print OUTFILE "\n        }";

                print OUTFILE "\n        if (FromLatitude >= SurfaceThreshold && m == 0)";
                print OUTFILE "\n        {";
                print OUTFILE "\n                MastAngle = MastAngle + MastRate; ";
                print OUTFILE "\n        /*Rate to raise mast when AUV is on water surface rate valur
from Uppaaal*/";
                print OUTFILE "\n                while (MastAngle>MaxAngle)   /*Take value of
MastAngle from Uppaal*/";
                print OUTFILE "\n                {";
                print OUTFILE "\n                        DevState__MastState = UP;";
                print OUTFILE "\n                        MastUp = 1;";
                print OUTFILE "\n                }";
                print OUTFILE "\n                glutPostRedisplay();";
                print OUTFILE "\n        }";

                print OUTFILE "\n}";
                $abortNumber++;
```

```
        }
# Generate the graphics of the AUV and undersea environment
        if ($Iterationnumber1 == 0 && $SteeringFunctionDone == 1 && $abortNumber == 1)
        {
                print OUTFILE "\nvoid reshape (int w, int h)";
                print OUTFILE "\n{";
                print OUTFILE "\n  glViewport (0, 0, (GLsizei) w, (GLsizei) h);";
                print OUTFILE "\n  glMatrixMode (GL_PROJECTION);";
                print OUTFILE "\n  glLoadIdentity ();";
                print OUTFILE "\n  gluPerspective(70.0, (GLfloat) w/(GLfloat) h, 10.0, 4.0);";
                print OUTFILE "\n  glMatrixMode(GL_MODELVIEW);";
                print OUTFILE "\n  glLoadIdentity();";
                print OUTFILE "\n  glTranslatef (0.0, 0.0, -5.0);";
                print OUTFILE "\n}";
#Associating mouse and keyboard related actions
                print OUTFILE "\nvoid keyboard (unsigned char key, int x, int y)";
                print OUTFILE "\n{";
                print OUTFILE "\nswitch (key) {";

                print OUTFILE "\n    case 'a':";
                print OUTFILE "\n                    {";
                print OUTFILE "\n                      i = 1;";
                print OUTFILE "\n                      glutIdleFunc(NULL);";
                print OUTFILE "\n           }";
                print OUTFILE "\n                    break;";

                print OUTFILE "\n    case 27:";
                print OUTFILE "\n                    exit(0);";
                print OUTFILE "\n          break;";
                print OUTFILE "\n    default:";
                print OUTFILE "\n          break;";
                print OUTFILE "\n          }";
                print OUTFILE "\n}";
                print OUTFILE "\nvoid mouse(int button, int state, int x, int y)";

                print OUTFILE "\n{";
                print OUTFILE "\n          switch(button)";
                print OUTFILE "\n          {";
                print OUTFILE "\n                    case GLUT_LEFT_BUTTON:";
                print OUTFILE "\n                    if (state == GLUT_DOWN && i == 0.0)";
                print OUTFILE "\n                    {";
                print OUTFILE "\n                              glutIdleFunc(ProcessPayload);";

                print OUTFILE "\n                    }          ";
                print OUTFILE "\n                                        break;";

                print OUTFILE "\n          }";
                print OUTFILE "\n}";


                print OUTFILE "\nFILE *SInput, *StInput, *SAngle;";
                print OUTFILE "\nint main(int argc, char** argv)";
                print OUTFILE "\n{";
                print OUTFILE "\n          glutInit(&argc, argv);";

                print OUTFILE "\n/*Open file for reading input*/";
```

```
                print OUTFILE '         SInput                                  =
fopen("C:\\Research\\Animation\\Temp\\Mission\\SteerInput.txt", "r");';
                print OUTFILE "\n       if( SInput == NULL )";
                print OUTFILE '         printf( "The file SteerInput.txt was not opened\n" );// File
failed to open';
                print OUTFILE "\n";
                print OUTFILE "\n       else";
                print OUTFILE '         printf( "The file SteerInput.txt was opened\n" );// File opened';
                print OUTFILE "\n";
                print OUTFILE "\n       fseek( SInput, 0L, SEEK_SET );";
                print OUTFILE "\n";
                print OUTFILE '         fscanf(SInput, "%d\n", &Steering);//Get the order is Steering
or not';
                print OUTFILE "\n";
                print OUTFILE '         fscanf(SInput, "%d\n", &SteerMode);//Mode of Steering';
                print OUTFILE "\n";
                print OUTFILE '         fscanf(SInput, "%f\n", &ToLatitude);//Get the destined
Latitude';
                print OUTFILE "\n";
                print OUTFILE '         fscanf(SInput, "%f\n", &ToLongitude);//Get the destned
Longitude';
                print OUTFILE "\n";
                print OUTFILE "\n       fclose(SInput);";
                print OUTFILE "\n";
                print OUTFILE '         printf("ToLat = %f, ToLong = %f\n",ToLatitude,
ToLongitude);';
                print OUTFILE "\n       /*Safety modules check for depth safety*/";
                print OUTFILE "\n       if (ToLatitude < -1.5)";
                print OUTFILE "\n       {";
                print OUTFILE "\n               ToLatitude = -1.5;";
                print OUTFILE "\n";
                print OUTFILE '         printf("ToLatitude value change to -1.5 as depth below that is
dangerous\n");';
                print OUTFILE "\n       }";
                print OUTFILE "\n       if (ToLatitude > 1.0)";
                print OUTFILE "\n       {";
                print OUTFILE "\n               ToLatitude = 1.0;";
                print OUTFILE "\n";
                print OUTFILE '                 printf("ToLatitude value cannot be more than 1.0 the
surface\n");';
                print OUTFILE "\n       }";

                print OUTFILE "\n";
                print OUTFILE '         StInput                                 =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "r");';
                print OUTFILE "\n";
                print OUTFILE '         fscanf(SInput, "%f\n", &FromLatitude);//Get the current
Latitude';
                print OUTFILE "\n";
                print OUTFILE '         fscanf(SInput, "%f\n", &FromLongitude);//Get the current
Longitude';
                print OUTFILE "\n";
                print OUTFILE '         printf("Lat = %f, Long = %f\n",FromLatitude,
FromLongitude);';
```

```
                print OUTFILE "\n       if (FromLatitude < -1.5)";
                print OUTFILE "\n       {";
                print OUTFILE "\n               FromLatitude = -1.5;";
                print OUTFILE "\n";
                print OUTFILE '\n               printf("FromLatitude value change to -1.5 as depth
below that is dangerous\n");';
                print OUTFILE "\n       }";

                print OUTFILE "\n       if (FromLatitude > 1.0)";
                print OUTFILE "\n       {";
                print OUTFILE "\n               FromLatitude = 1.0;";
                print OUTFILE '         printf("FromLatitude value cannot be more than 1.0 the
surface\n");';
                print OUTFILE "\n       }";
                print OUTFILE "\n       temp1 = FromLatitude - ToLatitude; ";
                print OUTFILE "\n       temp2 = FromLongitude - ToLongitude;";
                print OUTFILE "\n       temp3 = temp1*temp1 + temp2*temp2;";
                print OUTFILE "\n       squareroot = sqrt(temp3); ";
                print OUTFILE "\n";
                print OUTFILE '         printf("%f\n", squareroot);';
                print OUTFILE "\n";
                print OUTFILE '         SAngle                                                  =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "r");';
                print OUTFILE "\n";
                print OUTFILE '         fscanf(SAngle, "%f\n", &MastAngle);';
                print OUTFILE "\n       if (MastAngle > 0 && FromLatitude < 1.0)";
                print OUTFILE "\n       {";
                print OUTFILE "\n               MastAngle = 0.0;";
                print OUTFILE '         printf("Mast is not raised as not on surface\n");';
                print OUTFILE "\n       }";
                print OUTFILE "\n       fclose(SAngle);";
                print OUTFILE "\n       if (MastAngle > 0 && FromLatitude == 1.0)";
                print OUTFILE "\n       {";
                print OUTFILE "\n               MastAngle = 0.0;";
                print OUTFILE "\n";
                print OUTFILE '         printf("Lower Mast before going below surface of water\n");';
                print OUTFILE "\n       }";
                print OUTFILE "\n  glutInit(&argc, argv);";
                print OUTFILE "\n  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);";
                print OUTFILE "\n  glutInitWindowSize (500, 500); ";
                print OUTFILE "\n  glutInitWindowPosition (100, 100);";
                print OUTFILE "\n  glutCreateWindow (argv[0]);";
                print OUTFILE "\n  init ();";
                print OUTFILE "\n  glutDisplayFunc(display); ";
                print OUTFILE "\n  glutReshapeFunc(reshape);";
                print OUTFILE "\n  glutKeyboardFunc(keyboard);";
                print OUTFILE "\n  glutMouseFunc(mouse);";
                print OUTFILE "\n  glutMainLoop();";
                print OUTFILE "\n  return 0;";
                print OUTFILE "\n}";
                $Iterationnumber1++;
        }
}
```

5 Coordinator synthesis

Our goal is to automate the synthesis of mission coordinators (i.e. controllers at the topmost layer) for hierarchy based intelligent control architecture for AUVs. The interactions within the modules in hierarchical intelligent control architecture are complex. Synthesis of a coordinator for such a system is a challenging task as it requires careful monitoring of the inputs received and the outputs sent. The controller is a hybrid system with discrete states and continuous dynamics. The continuous dynamics are implemented as functions. The coordinators are a special case of hybrid system which only involve timing constraints known as timed automata.

Many systems belong to the group of hybrid systems such as logic based switching control systems, intelligent vehicle/ highway systems and chemical batch processes. A growing need for modeling, analysis and design of hybrid systems in practice has increased the efforts put in by researchers. Next we look into the different approaches to controller synthesis before we look into the details of our method.

Several approaches like game theory, supervisory control, and optimal control have been used to synthesize a controller. The supervisory control of discrete event system approach of Ramagde and Wonham [65] can also be said as the event feedback scheme. The plant generates events. The supervisor observes the events and then generates a control pattern based on a legal set of specifications. Other approaches have used state feedback control scheme [66] as shown in figure. The supervisor observes the plant states. At each step the supervisor generates a control pattern based on a given set of legal states to ensure no illegal state are reached.

The approach by Lygeros in [2], [62], [63], [64] is to design a hybrid controller by determining continuous control laws and conditions under which they satisfy the closed loop requirements. Then, a discrete design is constructed to ensure that these conditions are satisfied. Controller synthesis for a real time system is proposed by Asarin in [60]. The controller in [60] is synthesized based on a winning strategy for certain games defined by automata or timed automata. Another game theoretic approach proposed in [61] is used for constructing reliable controllers for arbitrarily large discrete systems. The controller is synthesized by finding a winning strategy for specific games defined by contracts. The discrete system model is an action system, and the requirement is a

temporal property. The game reduces to a competition between, the controller, and the plant, which try to prevent each other from achieving their respective goals. If the synthesis is possible, that is, if the controller has a way to enforce the required property, the process ends with finding the winning strategy of the controller, by propagating backwards the computed precondition of the plant, with respect to that property. This technique guarantees the correctness of the derived program.

5.1 Proposed Approach for coordinator synthesis

Automated synthesis of the coordinators promises reduction in time to develop and implement coordinators for underwater and aerial vehicles. It also improves modification and debugging capability.

Our goal in the automation of coordinators is to translate the higher level specifications and user inputs into sequence of actions to successfully execute the mission. The coordinators we synthesize are timed automata with timing constraints. The definitions of the automata built are as discussed in section.

We consider the requirement of three coordinators at the topmost level. The three coordinators are a sequential coordinator (implementing sequential control to execute a sequence of actions for a mission), a timed coordinator (implementing time critical missions) and a safety coordinator (implements safe execution of mission). These coordinators are synthesized based on user input and high level specification. The coordinators consist of a basic structure and a synthesized part. The basic structure implements control common to any kind of mission coordinator built. For example each and every coordinator needs to establish connection with the vehicle before requesting a mission. The synthesized part is developed based on the specific mission to be executed. For example a mission can be find the present location using a GPS or fire a missile.

5.2 Sequential coordinator

Sequential coordinator is used to coordinate the execution of sequence of actions involved in the successful execution of an untimed mission. The sequential coordinator is synthesized based on the inputs received and its response. Inputs received by the sequential coordinator can be requests made by the user i.e. the mission order or other coordinators at the same level or responses received from lower level or same level

coordinators. The algorithm consists of two parts the first part implements the basic structure and the second part implements the augmentation of new edges, guards, reset values and locations to the sequential coordinator. The simplest structure of the sequential coordinator is shown in Figure 52. The basic structure is the same for all the sequential coordinators which involves two different phases: Initialization phase, and Communication establishment phase. The mission specific structure contains mission order phase, and response phase.

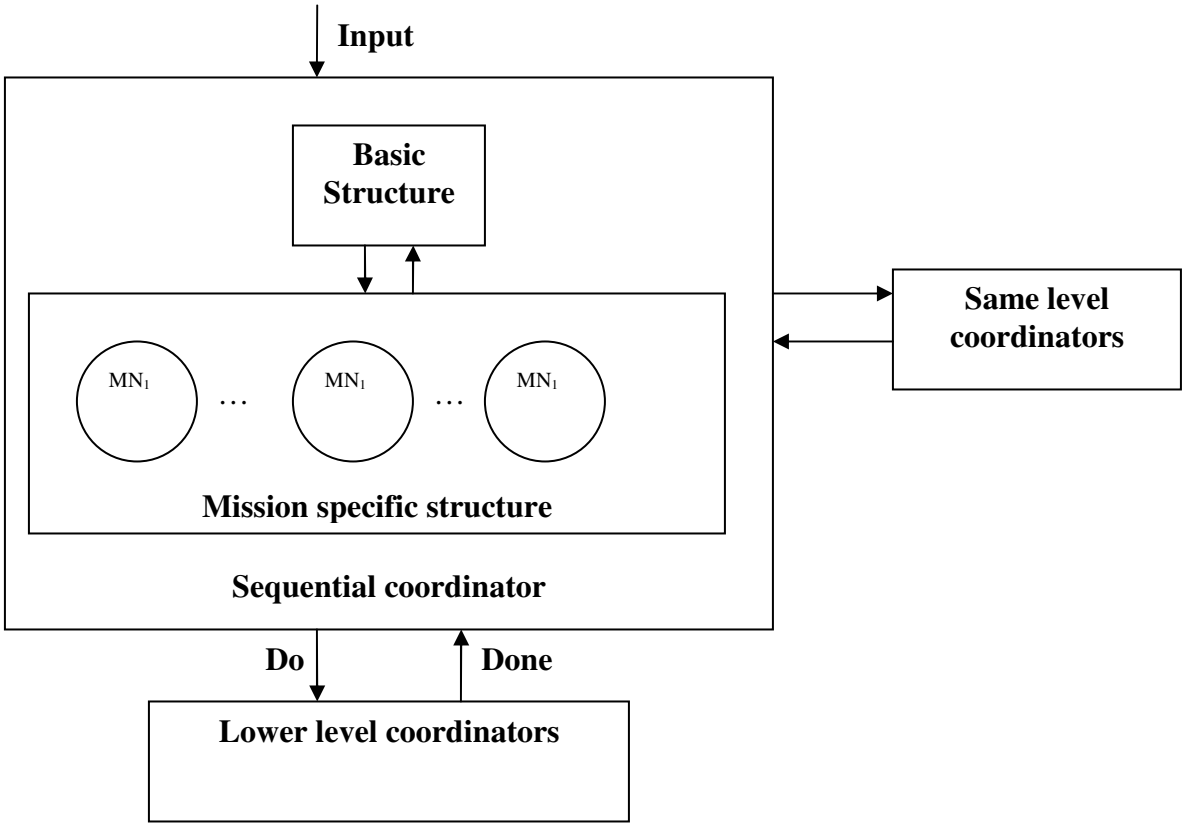


Figure 52: Structure of sequential coordinator

Algorithm:

Create five locations $l \in L$ and name them as ***Idle, WaitforVCComms, Run, Suspend*** and ***Endmission***. (control for any AUV needs all these states)

- Create an edge e_0 from ***Idle*** to ***WaitforVCComms*** (indicating transition to a state to wait to establish communication with the Vehicle)
    - Set event σ_{in} = *Init*

- Set guard condition $G^i(e_0) = t>=T$ where T is a constant time to initialize the system (for our case $T = 1$)
- Set reset condition $R(e_0) = \{t=0\}$

- Create an edge e_1 to ***Run*** state from ***WaitforVCComms*** if a connection with vehicle is established
  - Set event σ_{in} = *NewVCData*
  - Set guard condition $G^i(e_1) = t>=10$
  - Set reset condition $R(e_1) = \{t = 0, MissionTime = 0, Suspendable = 0\}$
- Create an edge e_2 from ***Run*** state to ***EndMission*** state
  - Set event σ_i = *Endmission*
  - For each $Controller_i \in Level_j$ where i= 1…n, j = 1 only
    - Set the guard condition on the edge $G^i(e_2) = (\cup Controller_k\text{->Idle})$ where k = 1,2…n, k != i checking the status of other controllers (0 meaning idle)
  - Set reset condition $R(e_2) = \{t = 0, Suspendable = 0, Idle = 0\}$ to indicate that all the coordinators are idle
- At ***EndMission*** state
  - Draw a self loop edge e_3
    - Set event σ_{in} = *OnSurface*
    - Set guard condition $G^i(e_3) = (Var_k < = SurfaceThreshold)$ where Var_k is k^{th} variable mapped to set of real numbers (indicating sensor value of depth) *SurfaceThreshold* indicates a constant value

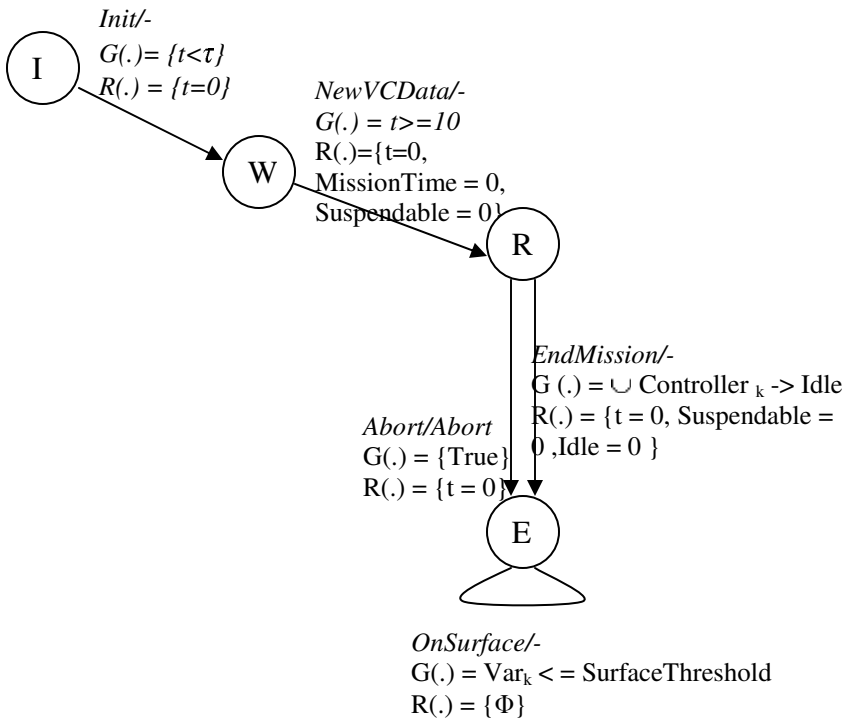


Figure 53: Basic structure for Sequential Coordinator

- **Start:** Get Mission order name **Or_n(<MissionName>, Prm)**.
- If mission name is obtained for the first time
  - Create a location $l \in L$ and name it **<MissionName>**
  - Draw an edge e_i from the ***Run*** state to **<MissionName>** state where i = j+1… n, where j is the number for the last edge that was drawn
    - Set the events as σ_{in}/σ_o = **<MissionName> /Do<MissionName>** command sent to the lower order controllers
  - Set the guard condition $G^j(e_i)$ = {Var_k = **"<MissionName>"}**
  - Set the reset condition R(e_i) ={Suspendable = (0 or 1), Idle = 0, t = 0}
  - If **Suspendable** = 1
    - Create an edge e_i from **<Mission Name>** state to ***Suspend*** state
      - Set event σ_{in} = *Suspend*
      - Set guard condition *G*(e_i) = *{True}*
      - Set reset condition *R*(e_i) = *{t = 0, Suspendable = 0}*
    - If connecting to the ***Suspend*** state for the first time

- Create a self loop e_i at the ***Suspend*** state
  - Set the event σ_{in} / σ_o = *Abort / Abort*
  - Set guard condition $G(e_i)$ = *{Var_k = !Suspended }*
  - Set reset condition $R(e_i)$ = *{Suspended = 1}*
- Create an edge e_i from ***Suspend*** state to ***Run*** state
  - Set event σ_{in} = *Resume*
  - Set guard condition $G(e_i)$ = *{True}*
  - Set reset condition $R(e_i)$ = *{Suspended = 0, Suspendable = 0, t = 0}*

- Draw an edge e_i from the **<Mission Name>** state to ***EndMission*** State
  - Set the σ_{in}/σ_o = *Abort / Abort*
  - Set guard condition $G(e_i)$ = *{True}*
  - Set reset condition $R(e_i)$ = *{t = 0}*
- Create an edge e_i from **<Mission Name>** state to ***Run*** State
  - Set event σ_{in} = **<MissionName>Done**
  - Set guard condition $G(e_i)$ = *{True}*
  - Set reset condition $R(e_i)$ = *{t = 0, Suspendable = 0}*

- Else if mission name is already there
  - Go to **Start** to get the name of the next mission
- End

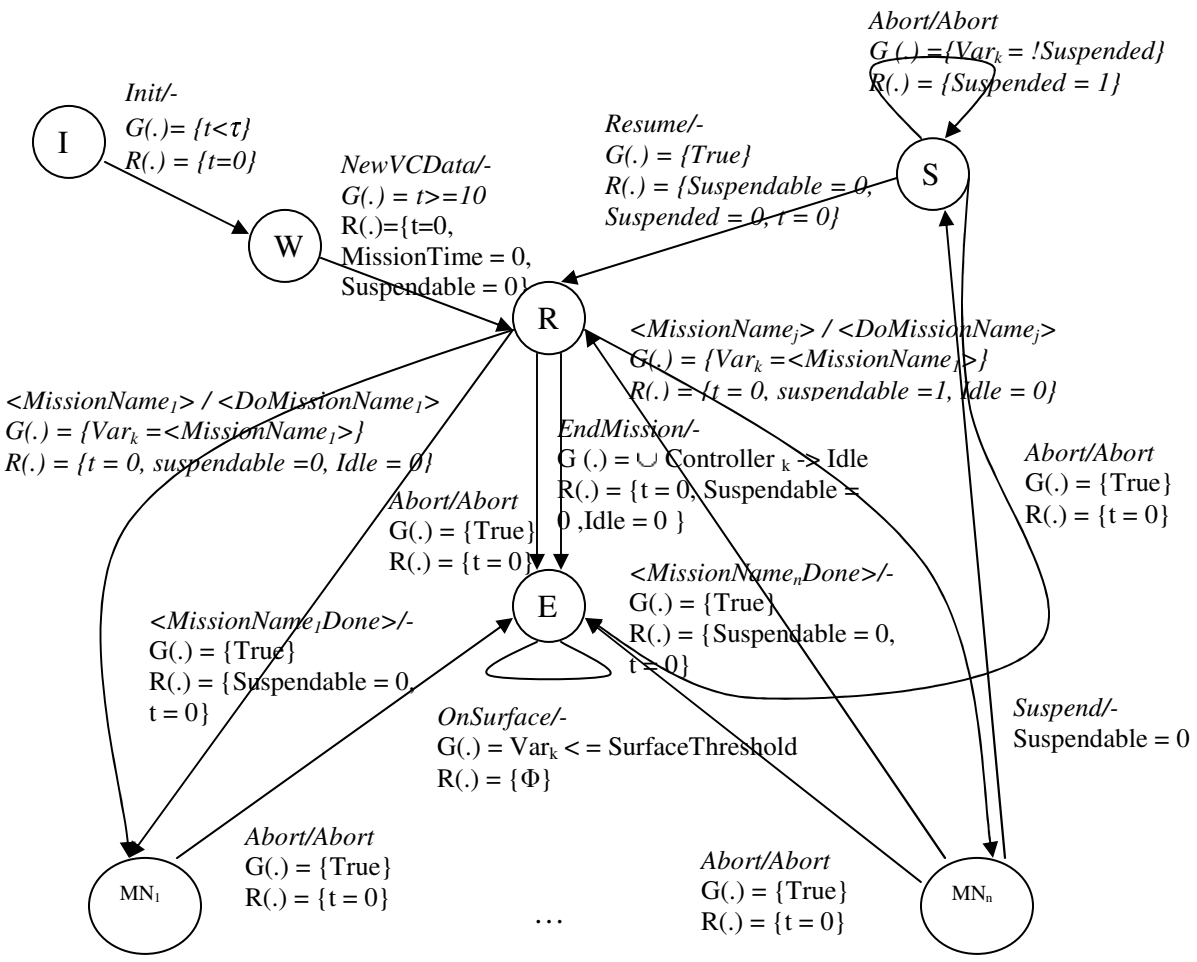


Figure 54 : Sequential coordinator

5.3 Timed coordinator synthesis

Timed coordinator is used to coordinate the execution of time critical mission. Timed critical mission involves execution of sequence of actions with timing constraints. The timed coordinator is synthesized based on the inputs received and its response. Inputs received by the sequential coordinator can be requests made by the user i.e. the mission order or other coordinators at the same level or responses received from lower level or same level coordinators. The algorithm till the label Start implements the basic structure and the remaining part implements the augmentation of new edges, guards, reset values and locations to the sequential coordinator.

Algorithm:

- Create seven locations $l \in L$ and name them as ***Idle, WaitForFirstTO, CheckOrders, Wait4Suspend, Check4Resume, Decide*** and ***End***.
- Draw an edge e_0 from ***Idle*** state to ***WaitForFirstTO*** state
    - Set event σ_{in} = *Init/-*
    - Set guard condition $G(e_0) = \{t >= \tau\}$ where τ is a constant
    - Set reset condition $R(e_0) = \{ \Phi \}$
- Draw an edge e_1 from ***WaitForFirstTO*** to ***CheckOrders***
    - Set event σ_{in} = *NewVCData/-*
    - Set guard condition $G(e_1) = \{True\}$
    - Set reset condition $R(e_1) = \{MissionTime = 0, t = 0, Done = 1\}$
- Draw an edge e_2 from ***CheckOrders*** state to ***End*** state
    - Set event σ_{in} = *EndMission*
    - Set guard condition $G(e_2) = \{True\}$
    - Set reset condition $R(e_2) = \{Idle = 0\}$
- Draw an edge e_3 from ***CheckOrders*** state to ***Decide*** state
    - Set event σ_{in} = *NewOrder*
    - Set guard condition $G^i(e)$ = strcmp(this->CurrTimedOrd->Name,"None") && TimedActions_get_MissionTime() >= this->CurrTimedOrd->Time && (!TimedActions_CheckSuspend(this)||this->SeqController->Idle||this->SeqController->Suspended)
    - Set reset condition $R(e_3) = \{Idle = 0\}$
- Draw an edge e_4 from ***CheckOrders*** to ***Wait4Suspend*** (indicating that the mission requires suspension of the other coordinators)
    - Set σ_{in} = *Suspend/ Suspend*
    - Set guard condition $G(e_4) = \{$ *strcmp(this->CurrTimedOrd->Name,"None") && TimedActions_get_MissionTime() >= this->CurrTimedOrd->Time && (TimedActions_CheckSuspend(this)&&this->SeqController->Suspendable && !this->SeqController->Idle) &&!this->SeqController->Suspended}*
    - Set reset condition $R(e_4) = \{t = 0, Idle = 0, Time2Suspend = 0\}$

- Create a loop e_5 at ***Wait4Suspend*** state
  - Set σ_{in} = *Suspend/ Suspend* (suspend the Sequential Coordinator)
  - Set guard condition $G^i(e_5)$ = !this->SeqController->Suspended
  - Set reset condition $R(e_5) = \{t = 0\}$
- Draw an edge e_6 from ***Wait4Suspend*** to ***Decide*** state
  - Set σ_{in} = *NewOrder*
  - Set $G^i(e_6)$ = this->SeqController->Suspended
  - Set reset condition $R(e_6) = \{\Phi\}$
- Draw an edge e_7 from ***Check4Resume*** to ***CheckOrders*** without *Resume* event
  - Set σ_{in} = *OrderComplete/-*
  - Set the $G^i(e_7)$ = !this->SeqController->Suspended || (TimedActions_get_MissionTime() >= this->CurrTimedOrd->Time && strcmp(this->CurrTimedOrd->Name,"None"))
  - Set the reset condition $R(e_7) = \{\Phi\}$
- Draw an edge e_8 from ***Check4Resume*** to ***CheckOrders***
  - Set σ_{in} = *OrderComplete /Resume*
  - Set $G^i(e_8)$ = this->SeqController->Suspended && (TimedActions_get_MissionTime() < this->CurrTimedOrd->Time || !strcmp(this->CurrTimedOrd->Name,"None"))
  - Set the reset condition $R(e_8) = \{\Phi\}$
- Draw an edge e_9 from each of the states (excepting ***Idle*** and ***WaitForFirstTO***) to ***End*** state
  - Set event σ_{in} / σ_o = *Abort/Abort*
  - Set guard condition $G^i(e_9) = \{True\}$
  - Set reset condition $R(e_9) = \{\Phi\}$
- **Start:** Get Mission order name **Or_n(<MissionName>, Prm)**.
- If mission name is obtained for the first time
  - Create a location $l \in L$ and name it **<MissionName>**
  - Draw an edge e_i from the ***Decide*** state to **<MissionName>** state where i = j+1…n where j is the number for the last edge drawn

        - Set σ_{in} / σ_o = **<MissionName>/ Do<MissionName>** sent to lower level controllers
        - Set guard condition $G(e_i) = \{CurrentOrder = <MissionName>\}$
        - Set reset condition $R(e_i) = \{\Phi\}$
    - Draw an edge e_i from the **<MissionName>** state to ***End*** State
        - Set σ_{in} / σ_o = *Abort*
        - Set guard condition $G(e_i) = \{True\}$
        - Set reset condition $R(e_i) = \{\Phi\}$
    - Create an edge e_i from **<MissionName>** state to ***Check4Resume*** State
        - Set σ_{in} = **<MissionName>Done** signal
        - Set guard condition $G(e_i) = \{True\}$
        - Set reset condition $R(e_i) = \{\Phi\}$
- Else if mission name is already there
    - Go to **Start** to look for the next order
- End

5.4 Safety Coordinator synthesis

Safety by definition is the freedom from danger, damage or risk. Thus the goal of a safety coordinator is to prevent the vehicle from taking actions which might damage the vehicle. The safety coordinator monitors the different parameters involved in the missions ordered by mission coordinators, the proper functioning of the components of the vehicle and the environment surrounding the vehicle. So a safety coordinator basically is an observer which acts only when the operations lead to unsafe state. When the safety coordinator finds that a mission prompts execution of an unsafe action it tries to correct the action and make it safe. If the safety coordinator is not able to make the mission safe it aborts the mission. For example if a mission commands the vehicle to go to a depth of 500ft and the present safe depth is only 200ft the safety coordinator changes the depth to 200ft. If the safety coordinator is able to correct it the mission is carried out or else it aborts the mission. We here list a set of safety issues a safety coordinator should satisfy.

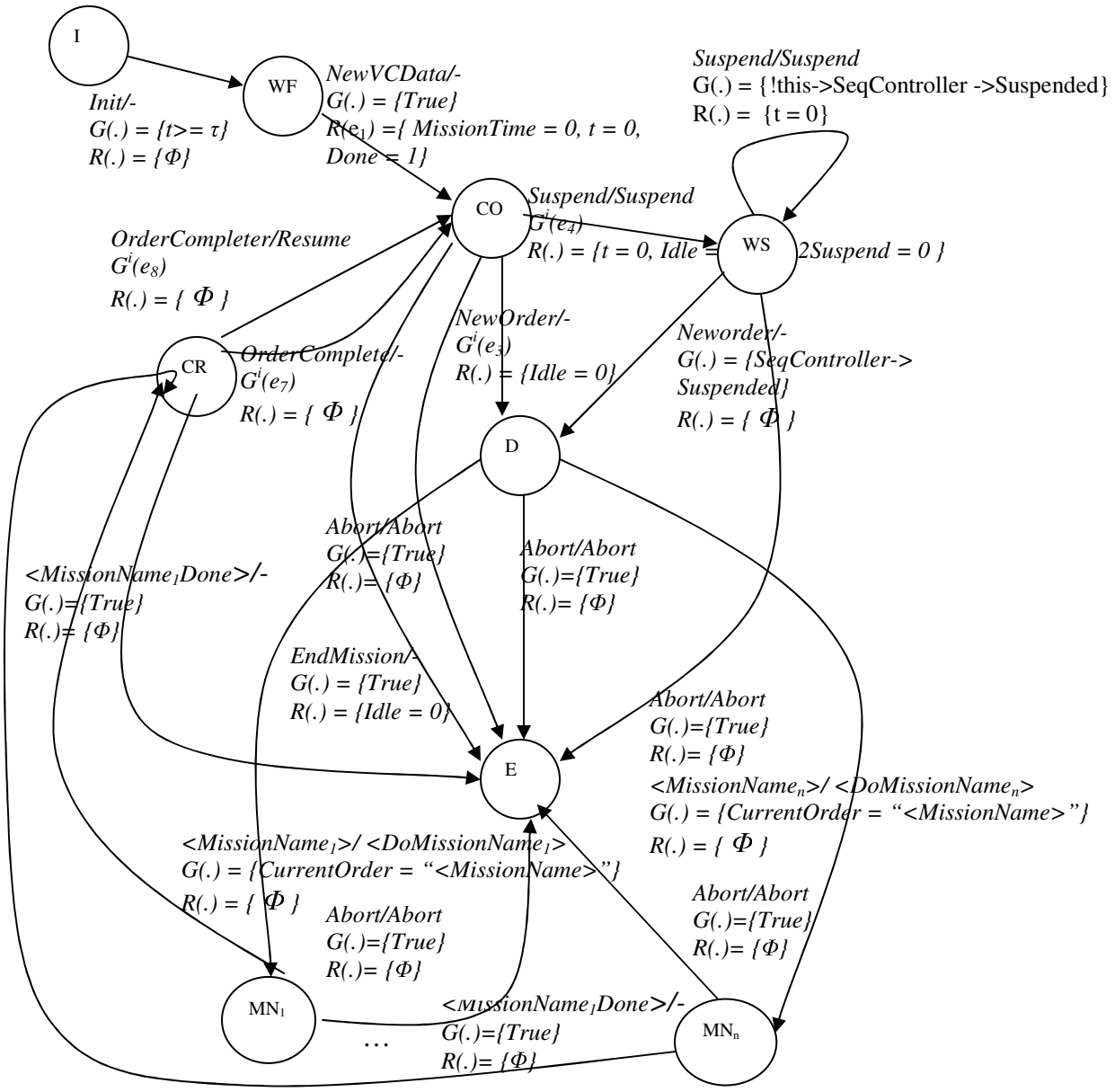


Figure 55 : Timed coordinator

The safety issues which a safety coordinator for an AUV should take care of are as listed below.

1. *Water depth safety* monitoring should check the altitude of the vehicle from the bottom of the sea and thus prevent the vehicle from hitting the bottom of the sea.

2. *Obstacle avoidance safety* should monitor the presence of obstacles which might be other vehicles, or mountains under sea and prevent collision of the AUV with the obstacle.

3. *Device functioning safety* should monitor the functioning of the different critical components which constitute an AUV. Critical components are those components malfunctioning of which might lead to damage of vehicle or undesirable situation like AUV stuck at the bottom of the sea due to battery failure.

All these safety issues can be modeled as constraints within a hybrid system as has been done for the survey AUV built at ARL. The constraints are the guard conditions which prompt the transition from one state to other depending upon the situation.

The coordinators synthesized here work together to successfully execute a mission. It is shown next. Given a mission the coordinators communicate among each other to successfully complete a mission.

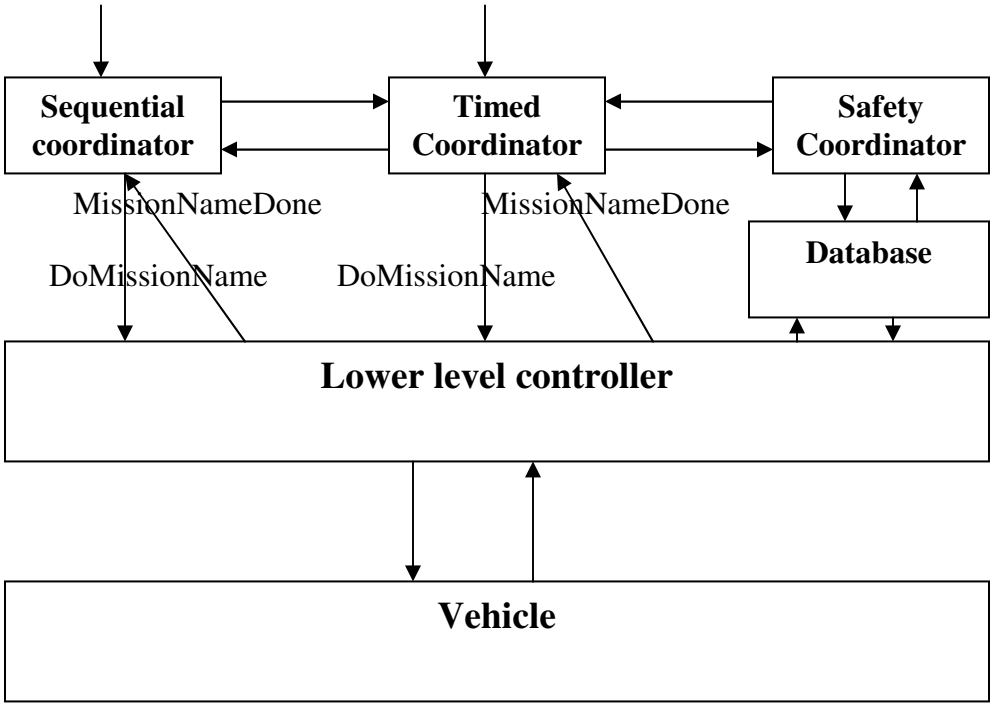


Figure 56: The complete structure

Here we are concerned with the successful execution of the mission as ordered by the highest level controllers which we have automatically synthesized.

The analysis is provided based on the interactions between the modules shown in Figure 56 and the detailed modules of the sequential (Figure 54) and timed coordinator (Figure 55).

Both the coordinators are initialized first. During initialization the sequential coordinator establishes contact with the vehicle and the terminal from which mission orders are received.

When new order is received both the coordinators transition to the state at which they become ready to execute a mission. If it's an untimed mission the sequential coordinator accepts the input and sends **<DoMissionName>** (Figure 56) to the lower level controllers. Once the mission is successfully executed the sequential coordinator receives **<MissionNameDone>** (Figure 56) from the lower level controllers**.** Then the SC considers the next order in queue and passes control to the concerned lower level controller. If due to some malfunctioning the mission needs to be terminated an *abort* signal is received by the sequential coordinator from the lower level controllers involved in the mission. The sequential coordinator then broadcasts the *abort* signal (Figure 54) and terminates the execution of all other missions. If there are no more orders in the queue the SC checks for the status of the TC. If the TC is idle SC sends *EndMission* and transitions to the ***EndMission*** state (Figure 54).

If a timed mission is received then the timed coordinator checks whether the execution of the present mission needs the suspension of the sequential coordinator or not (these constraints are guard conditions on edges). If TC needs to suspend SC, TC sends the *suspend* signal to SC (Figure 55). If the mission which SC is executing is suspendable then SC synchronizes with the event *suspend* and transitions to the ***Suspend*** state (Figure 54). When SC is suspended TC sends the order as **<DoMissionName>** to the lower level controllers (Figure 56). The lower level controllers respond back with the **<MissionNameDone>** event to the TC when the mission is completed (Figure 56). TC then finds the next order in queue and either resumes the SC or keeps it suspended or keeps it unsuspended (Figure 55).

The safety coordinator keeps checking the parameters from the mission and sensor values of the AUV from the common database to safely execute a mission (Figure 56).

This way all the coordinators interact with each other and complete the execution of a mission order successfully.

Lemma 5.1: *(Given no Abort event) for all the orders there exist a response from the lower level controller which completes the mission successfully.*

Proof: *Lemma 5.1* can be reduced to the expression

$$\forall m \in M((\exists \sigma_i \mid H_i^j \xrightarrow{\sigma_i} H_k^p) \,\&\, (\exists \sigma_k \mid H_k^p \xrightarrow{\sigma_k} H_i^j)) \text{ --- 5.1}$$

Equation 5.1 states that for all missions there exist an event to pass control to the concerned commanded controllers as well as there exist an event to let the commanding controller know the completion of the mission.

From the coordinator synthesis algorithms we find that for the missions there exist a method to pass control from the higher level coordinator i.e. S.C or T.C. to the lower level coordinator which is to synchronize on common events <*DoMissionName*>. Thus we can express it as

$\forall m \in M(\exists \sigma_i \mid H_i^j \xrightarrow{\sigma_i = <DoMissionName>} H_k^p)$ where i indicates the subsystem at level j, k indicates the subsystem at level p, $j>p$ indicating that level j is at a higher level than level k --- 5.2

From the algorithms we find that each of the lower level coordinators respond back to a <*DoMissionName*> by a <*MissionNameDone*> event sent to the higher level controller.

$$\forall m \in M(\exists \sigma_k \mid H_k^p \xrightarrow{\sigma_k = <MissionNameDone>} H_i^j) \text{ --- 5.3}$$

Equation 5.2 and 5.3 together state that for each and every mission to be executed there exist an event to pass the control to the concerned controller as well as there exist a response which tells the higher level coordinator that the mission has been successfully executed. Thus equation 5.1 holds so does *Lemma 5.1*.

Lemma 5.2: *If the order is an Abort event it terminates the mission.*

Proof: The above *Lemma* can be reduced to the expression

$$\forall l \in L \forall \sigma = Abort \exists E \mid l \xrightarrow{\sigma = Abort} l_{final} \text{ --- 5.4}$$

The expression states that for all locations belonging to a set of locations and for all events which are *Abort* events there exist an edge in which a transition occurs from the present location, the source to the target location, the final state. If the above expression holds for the coordinators synthesized by the algorithm then ***Lemma 5.2*** holds.

From the coordinator synthesis algorithm we find that there are statements which implement edges with *Abort* events from the ***<MissionName>*** locations to the ***Endmission*** location.

$$\textbf{<MissionName>} \xrightarrow{\sigma=Abort} \textbf{Endmission}$$

Thus equation 5.4 holds and so does *Lemma 5.2*.

Lemma 5.3: *Timed as well as untimed missions can be successfully executed by the timely coordination between the Timed and Sequential coordinator.*

Proof: The above *Lemma 5.3* can be reduced to the expression $\forall m \in M \exists \sigma \mid (TC \xrightarrow{\sigma} SC)$ ---5.5.

The expression states that for all missions there exist a coordination event between the Timed Coordinator and the Sequential Coordinator for succesfull completion of both timed and untimed missions. If the above expression is satisfied by the coordinator synthesis algorithm then *Lemma 5.3* holds.

In the Sequential Coordinator synthesis algorithm we find statements dealing with creation of edges on value of Suspendable =1 and σ = *Suspend.* In the Timed Coordinator synthesis we find the implementation of edge e_4 which implements sending σ = *Suspend* to the Sequential Coordinator.

The above statements reduce to $\exists \sigma = Suspend \mid TC \xrightarrow{\sigma=Suspend} SC$ --- 5.6

The above expression states that there exist an event which aids TC to Suspend SC for execution of timed events.

From both the TC and SC synthesis algorithms we find statements implementing σ = *Resume* which helps in resuming the suspended SC. This statement reduces to the expression.

$\exists \sigma = \text{Re}\, sume \mid TC \xrightarrow{\sigma=\text{Re}\, sume} SC$ --- 5.7

From equations 5.6 and 5.7 we find that there exist methods of coordination between both the TC and the SC to execute timed as well as untimed missions. Thus equation 5.5 holds, so does the *Lemma 5.3*.

Lemma 5.4: *Given a mission*

(a) If no abort occurs during mission operation, then the mission will be successfully completed.

(b) If an Abort occurs during the mission, then the mission is terminated

(c) Timed and Sequential coordinator can coordinate among each other by suspending the other if required for execution of a mission.

Proof: *Lemma 5.1-5.3* prove *Lemma 5.4*.

6 Conclusion and future work

We have proposed hierarchical hybrid mission control architecture for AUVs. The architecture has been successfully implemented at ARL. The modular approach to execute a mission breaks down complex missions into simple modules which aids in easy design and implementation of the architecture. The present architecture is also not strictly priority driven. A priority driven architecture with timed and untimed missions strictly separated would lead to lesser missions being expired and give efficient performance. A priority driven system with higher priority for timed missions would look into all the timed and untimed missions in a queue and execute timed missions before the untimed ones if there is a possibility of missing the successful completion of timed missions.

A real-life, complex, and hierarchically structured hybrid control system has been verified using our bottom up approach. The advantage of bottom-up approach is reduction of complexity, and also if an error is detected in a certain module, the lower level modules do not need to be revised. The current approaches typically consider reachability properties for verification, but through our modular bottom up approach we are able to analyze the correctness of the entire controller. As far as logical correctness is concerned, we verified 12 different modules against a total of 148 queries; the table shows the number of queries and the name of the module. The verification confirms the correctness of designed modules for progress.

S.No.	Name of the subsystem	No. Of Queries
1	Steering	3
2	Loiter	22
3	Rendezvous	8
4	Payload	4
5	Pause	2
6	Launcher	4
7	GPSFixer	12
8	DeviceCommander	4

9	WaypointNavigator	26
10	Sequential Coordinator	34
11	Timed Coordinator	25
12	Safety Coordinator	4

The problem of complexity still exists if a modular design is not performed, and properties to be verified are not dependent on behaviors of small sub-collection of modules, rather the entire set of modules. In this present work we verified logical correctness of the missions executed. The correctness of function-calls is another issue not addressed here. Function-call verification will include the verification of the whole hybrid system. The present architecture can be extended to multiple underwater-vehicles.

We have simulated a hierarchically organized mission controller architecture using a bottom up approach to conversion from coordinator modules to OpenGL code. The simulation involved all the coordinators involved in a specific operation and thus strengthened the correctness of the model. The simulation proved the feasibility of building a simulation tool for a mission driven AUV. The simulation tool can be further developed and generalized to be used for any kind of mission (other than just survey) for an AUV. The simulation tool can also be enhanced to get real time sensor information as feedback and then take actions accordingly. The simulation tool can be further advanced so as to implement the complex mathematical models involved and give a much more accurate and attractive result.

Finally we have designed automated synthesis of coordinators. These synthesis algorithms need to be implemented. The coordinators synthesized (with modification of safety coordinator) can be used in future for hierarchical hybrid mission control architecture for aerial vehicles as well.

7 References

[1] R. Horowitz and P. Varaiya, "Control design of an automated highway system," Proc. IEEE, vol. 88, pp. 913–925, July 2000.

[2] J. Lygeros, D. N. Godbole, and S. Sastry, "Verified hybrid controllers for automated vehicles," IEEE Trans. Automat. Contr., vol. 43, pp. 522–539, Apr. 1998.

[3] P. Varaiya, "Smart cars on smart roads: Problems of control," IEEE Trans. Automat. Contr., vol. 38, no. 2, pp. 195–207, 1993.

[4] C. Livadas, J. Lygeros, and N. Lynch, "High-level modeling and analysis of the traffic alert and collision avoidance system (TCAS)," Proc. IEEE, vol. 88, pp. 926–948, July 2000.

[5] J. Lygeros, G. J. Pappas, and S. Sastry, "An approach to the verification of the Center-TRACON Automation System," in Hybrid Systems: Computation and Control, T. Henzinger and S. Sastry, Eds. Berlin, Germany: Springer-Verlag, 1998, vol. 1386, Lecture Notes in Computer Science, pp. 289–304.

[6] C. Tomlin, G. J. Pappas, and S. Sastry, "Conflict resolution for air traffic management: A study in muti-agent hybrid systems," IEEE Trans. Automat. Contr., vol. 43, no. 4, pp. 509–521, April 1998.

[7] A. Balluchi, L. Benvenuti, M. DiBenedetto, C. Pinello, and A. Sangiovanni-Vincentelli, "Automotive engine control and hybrid systems: Challenges and opportunities," Proc. IEEE, vol. 88, pp. 888–912, July 2000.

[8] K. R. Butts, "Analysis needs for automotive powertrain control," presented at the 7th Mechatronics Forum Int. Conf., Atlanta, GA, Sept 2000.

[9] D. Pepyne and C. Cassandras, "Optimal control of hybrid systems in manufacturing," Proc. IEEE, vol. 88, pp. 1108–1123, July 2000.

[10] S. Engell, S.Kowalewski, C. Schulz, and O. Stursberg, "Continuous discrete interactions in chemical processing plants," Proc. IEEE, vol. 88, pp. 1050–1068, July 2000.

[11] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee, "Modular specification of hybrid systems in CHARON," in Hybrid Systems: Computation and Control, N.

Lynch and B. H. Krogh, Eds. New York: Springer-Verlag, 2000, vol. 1790, Lecture Notes in Computer Science.

[12] M. Song, T.-J. Tarn, and N. Xi, "Integration of task scheduling, action planning, and control in robotic manufacturing," Proc. IEEE, vol. 88, pp. 1097–1107, July 2000.

[13] O. Maler and S. Yovine, "Hardware timing verification using KRONOS," in Proc. 7th Conf. Computer-Based Systems and Software Engineering, 1996.

[14] IEEE Trans. Automat. Contr., vol. 43, no. 4, Apr. 1998. Special Issue on Hybrid Systems.

[15] Automatica, vol. 35, no. 3, Mar. 1999. Special Issue on Hybrid Systems.

[16] R. Alur, T. A. Henzinger, and E. D. Sontag, Eds., Hybrid Systems III. New York: Springer-Verlag, 1996, vol. 1066, Lecture Notes in Computer Science.

[17] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, Eds., Hybrid Systems II. New York: Springer-Verlag, 1995, vol. 999, Lecture Notes in Computer Science.

[18] R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, Eds., Hybrid Systems. New York: Springer-Verlag, 1993, vol. 736, Lecture Notes in Computer Science.

[19] T. Henzinger and S. Sastry, Eds., Hybrid Systems: Computation and Control. New York: Springer-Verlag, 1998, vol. 1386, Lecture Notes in Computer Science.

[20] O. Maler, Ed., Hybrid and Real-Time Systems. New York: Springer-Verlag, 1997, vol. 1201, Lecture Notes in Computer Science.

[21] F. W. Vaandrager and J. H. van Schuppen, Eds., Hybrid Systems:Computation and Control. New York: Springer-Verlag, 1999, vol. 1569, Lecture Notes in Computer Science.

[22] www.teja.com

[23] www.uppaal.com

[24] Rajeev Alur, Thomas A. Henzinger, Gerardo Lafferriere, And George J. Pappas *Discrete Abstractions of Hybrid Systems*. Proceedings of the IEEE, 88, July 2000.

[25] J.Albus and R. Quintero, toward a refernce model architecture for real-time intelligent control systems (artics.), ASME Press series, 3, 1990.

[26] R.A.Brooks A robust layered control system for mobile robot. IEEE Journal Of Robotics And Automation 2(1):14-23, March 1986.

[27] Daniel Simon, Eve-Coste Maniere, Roger Pissard (INRIA BP93), Vincent Rigaud, Michel Perrier, Alexis Peuch (IFREMER BP 330) France, A reactive approach to underwater vehicle control: the Mixed ORRCAD/PIRAT programming of the VORTEX vehicle.

[28] J.Lygeros, Hierarchical hybrid control of large scale systems. PhD Thesis department of Electrical Engineering, University Of California Berkeley, 1996.

[29] Dov M. Gabbay, et al Temporal Logic: Mathematical Foundations and Computational Aspects: Volume 2.

[30] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi, 1995, "A user guide to HyTech", *Proceedings of the First International Workshop on Tools and Algorithms for the Construction and Analysis of Systems* (TACAS '95), Lecture Notes in Computer Science 1019, Springer-Verlag, 41-71.

[31] Kim G. Larsen and Paul Pettersson, 2000, "Timed and Hybrid Systems in UPPAAL2k", *MOVEP'2k : Modeling and Verification of Parallel Processes*, Nantes, France.

[32] M. M. Gupta and N. K. Sinha, Eds., *Intelligent Control: Theory and Applications*. Piscataway, NJ: IEEE, 1996.

[33] C. J. Harris, Ed., *Advances in Intelligent Control*. New York: Taylor & Francis, 1994.

[34] P. J. Antsaklis and K. M. Passino, Eds., *An Introduction to Intelligent Autonomous Control*. Norwell, MA: Kluwer, 1993.

[35] D. A. White and D. A. Sofge, Eds., *Handbook of Intelligent Control*.New York: Van Nostrant Reinhold, 1992.

[36] G. N. Saridis, "Analytical formulation of the principle of increasing precision with decreasing intelligence for intelligent machines," *Automatica*, vol. 25, no. 3, pp. 461–467, 1989

[37] A. Meystel, "Nested hierarchical control," in *An Introduction to* Intelligent and Autonomous Control. Norwell, MA: Kluwer, 1993, pp. 129–161.

[38] B. P. Zeigler and S. Chi, "Model-based architecture concepts for autonomous systems design and implementation," in An Introduction to Intelligent and Autonomous Control. Norwell, MA: Kluwer, 1993, pp. 57–78.

[39] L. Acar and U. Ozguner, “Design of structure-based hierarchies for distributed intelligent control,” in An Introduction to Intelligent and Autonomous Control. Norwell, MA: Kluwer, 1993, pp. 79–108.

[40] J. S. Albus, “Data storage in the cerebellar model articulation controller (CMAC),” J. Dyn. Syst., Meas. Contr., vol. 93, pp. 228–233, 1975.

[41] J. S. Albus “A new approach to manipulator control: The cerebellar model articulation controller (CMAC),” J. Dyn. Syst., Meas. Contr., vol. 93, pp. 220–227, 1975.

[42] “A reference model architecture for intelligent systems design,” in An Introduction to Intelligent and Autonomous Control. Norwell, MA: Kluwer, 1993, pp. 27–56.

[43] R. Kumar and V. K. Garg, Modeling and Control of Logical Discrete Event Systems. Norwell, MA: Kluwer, 1995.

[44] A. H. Levis, “Modeling and design of distributed intelligence systems,” in An Introduction to Intelligent and Autonomous Control. Norwell, MA: Kluwer, 1993, pp. 109–128.

[45] Ratnesh Kumar and James A. Stover, A Behavior-Based Intelligent Control Architecture with Application to Coordination of Multiple Underwater Vehicles, IEEE transactions on systems, man, and cybernetics—part a: systems and humans, vol. 30, no. 6, november 2000.

[46] Arkin, R. C. Behavior-Based Robot Navigation for Extended Domains. *Adaptative Behavior*, 1992, 1 (9),pp. 201-225.

[47] Warren, C. W. A technique for autonomous underwater vehicle route planning. In: *IEEE Journal of Oceanic Engineering,* 1990 15 (3), 199-204.

[48] Steels, L. The PDL reference manual. VUBAI Lab, Memo 92-5. Brussels, Belgium 1992.

[49] Maes, P. Situated Agents Can Have Goals. In: *Robotics and Automation Systems,* 6 p. 49-70, 1990.

[50] M. Carreras, J. Batlle and P. Ridao Hybrid Coordination of Reinforcement Learning-based Behaviors for AUV Control

[51] Sutton, R. and Barto, A. *Reinforcement Learning, anintroduction*. MIT Press, 1998.

[52] Watkins, C.J.C.H., and Dayan, P. Q-learning. *MachineLearning*, 8:279-292, 1992.

[53] Maes, P. and Brooks, R. Learning to coordinate behaviors. In *Proceedings of the Eighth AAAI*, pages 796-802. Morgan Kaufmann, 1990.

[54] Gachet, D., Salichs, M., Moreno, L. and Pimental, J. Learning Emergent tasks for an Autonomous Mobile Robot, *Proceedings of the International Conference on Intelligent Robots and Systems (IROS '94)*, Munich, Germany, September, pp. 290-97, 1994.

[55] Mahadevan, S. and Connell, J. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311- 365, 1992.

[56] Shackleton, J. and Gini, M. Measuring the Effectiveness of Reinforcement Learning for Behavior-based Robots. *Adaptive Behavior*, 1997.

[57] Touzet, C. Neural reinforcement learning for behavior synthesis. In*: Robotics and Autonomous Systems*, 22, 251-281, 1997.

[58] Stefan B. Williams, Paul Newman, Gamini Dissanayake, Julio Rosenblatt, Hugh Durrant-Whyte, A decoupled, distributed AUV control architecture, Australian Centre for Field Robotics University of Sydney NSW 2006, Australia.

[59] Pere Ridao, Josep Forest, Jordi Freixenet, and Joan batlle, Towards a distributed object oriented control architecture for autonomy.

[60] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid System II*, volume 999 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995

[61] Ralph-Johan Back, Cristina Cerschi Seceleanu, Contracts and Games in Controller Synthesis for Discrete Systems, 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04) 05 24 - 05, 2004 Brno, Czech Republic

[62] Lygeros, J.; Godbole, D.N.; Sastry, S. "A game-theoretic approach to hybrid system design", IN: *Hybrid Systems III. Verification and Control*, New Brunswick, NJ, USA, 22-25 Oct. 1995). Edited by: Alur, R.; Henzinger, T.A.; Sontag, E.D. Berlin, Germany: Springer-Verlag, 1996. p. 1-12.

[63] C. Tomlin, G. Pappas, and S. Sastry, ``Conflict resolution for air traffic management: A case study in multi-agent hybrid systems," tech. rep., UCB/ERL M97/33, Electronics Research Laboratory, University of California, Berkeley, 1997. To appear in the IEEE Transactions on Automatic Control, Special Issue on Hybrid Systems, April 1998.

[64] Lygeros, J.; Tomlin, C.; Sastry, S. "Multiobjective hybrid controller synthesis", IN: Hybrid and Real- Time Systems. International Workshop, *HART'97. Proceedings*, Grenoble, France, 26-28 March 1997). Edited by: Maler, O. Berlin, Germany: Springer-Verlag, 1997. p. 109-23.

[65] Ramagde, P.J., Wonham W.M..“The control of Discrete Event Systems”, Proceedings IEEE, 1989, Vol. 77, No. 1, pp. 81-98.

[66] Holloway, L.E., Krogh, B.H., Giua, A.. “A survey of Petri Net Methods for Controlled Discrete Event Systems,” Journal of Discrete Event Systems, 1997, Vol. 7, No. 2, pp. 151-190.

[67] Intellimotion, “Keeping up with California PATH research in Intelligent Transportation Systems”, Vol. 5 No. 1, 1996.

[68] Daniel Wiesmann, Duke Lee, Carmma user guide California PATH research in Intelligent Transportation Systems.

[69] Datta N. Godbole, John Lygeros, and Shankar Sastry, “Hierarchical Hybrid Control: a Case Study”, LNCS 999, June 1995.

[70] Jean Della Dorat, Aude Maignant, Mihaela Mirica-Ruse, Sergio Yovine[#] *Hybrid Computation*, *LMC-IMAG, 51 rue des Mathématiques**,** 38041 Grenoble Cedex9, France, [#]VERIMAG, Centre Equation, 2, Ave de Vignate**,** 38610 Gieres, France.

[71] http://www.cs.utexas.edu/users/moore/best-ideas/nqthm/

[72] R. Alur, C. Courcoubetis, T.A. Henzinger, P.-H. Ho. “Hybrid Automata: An algorithmic approach to the specification and verification of hybrid systems”**.** In *Hybrid Systems,* LNCS 736, pp. 209-229, 1993.

[73] R. David, "Modeling of Hybrid Systems Using Continuous and Hybrid Petri Nets," *Proc. of Conf. on Petri Nets and Performances Evaluation* (Saint Malo, France), pp. 47-58, June, 1997.

[74] D. Liberzon, and A.S. Morse, "Basic problems in stability and design of switched systems", *IEEE Control Systems Magazine, 19(5):59-70, October 1999.*

[75] Jan Lunze, "What is a hybrid system", Lecture Notes in Control and Information Sciences 279, 2002.

[76] Rajeev Alur, Thomas A. Henzinger, Pei-Hsin Ho, Automatic Symbolic Verification of Embedded Systems**,** IEEE Transactions on Software Engineering Volume 22 , Issue 3 (March 1996) Pages: 181 – 201, 1996.

[77] http://www-vrimag.imag.fr/TEMPORISE/kronos/

[78] http://spinroot.com/spin/whatispin.html

[79] http://hol.sourceforge.net/

[80] http://pvs.csl.sri.com/

[81] http://www.cis.ksu.edu/~allen/lambda.html

[82] http://gtps.math.cmu.edu/tps.html

[83] L.E.Holloway, Xiaoyi Guan, Ranganathan Sundaravadivelu, and Jeff Ashley,Jr. Automated Synthesis and Composition of Taskblocks for control of manufacturing systems, IEEE Trans. on Systems, Man, and Cybernetics, Vol30, No.5, October 2000.

[84] Marc Carreras i Pérez, Joan Batlle I Grabulosa An overview of Behavioural-based Robotics with simulated implementations on an Underwater Vehicle.

[85]

Appendix A: Commands for the underwater vehicle for search

Abort: This command is given to terminate an operation or procedure before completion, if some other higher priority operation needs to be taken care of or the present job doesn't need to be done.
DeviceDone: This response is sent by the Device Commander when a device required for an operation has been set.
GoToEndMission: This command is sent to indicate that a mission has been accomplished so all the operations are terminated.
GoToRendezvous: This command is sent to go to the desired meeting point.
GPSFixDone: This is response sent by the GPSFixer once the global positioning system finds the position.
Init: This command is sent by all the modules as an initializing command after which it transitions to the Idle state from the Start state and becomes ready for operation.
Launch: This command is sent to activate the Launcher module.
LaunchDone: This is response to the command launch sent once the function of the launcher is done with.
MastDown: This command is sent to the vehicle controller mast to lower the mast.
MastUp: This command is sent to rise the mast.
NewVCData: This is the data obtained by the Vehicle controller sensors.
OnSurface: This command is sent to indicate that the underwater sea vehicle has reached the water surface.
PayLoadDone: This is response given by the PayLoad module to indicate that payload has been delivered.
ProcessPayLoad: This command is sent to start the processing of the payload.
ProcessWP: This command is sent to process the direction of the vehicle based on the way point (it gives the coordinates of a point) it needs to go to.
RendezvousDone: This is response given to indicate that the rendezvous with other underwater vehicles has been done.
Resume: This command is given to resume operation after suspension.
SetDevice: This command is given to the Device commander to set a concerned device for a certain operation.
SuspendBehavior: This command is given to suspend a behavior.
TakeGPSFix: This command is given to find the global position of the vehicle.
Update: This command updates the present location of the vehicle.
Wait: This command is given to wait for sometime before staring on or resuming some operation.
WaitDone: This response is sent to indicate that waiting period is over now operation can be resumed.
WPDone: This response indicates that way point or location has been found.
AltitudeOK: This respone says that the depth to which the vehicle needs to go to is safe.
AltitudeSafety: This command checks whether the altitude level to go to is safe or not.
DeliverPayLoad: This command tells to deliver the payload.
GoToLoiter: This command tells the vehicle to go to start loitering.
LightOff: This command checks whether light is switched off
Loiter: This command is given to the vehicle to loiter in its surrounding area.

LoiterDone: This response indicates that loitering is done.
Steer: This command is sent to steer the vehicle to the desired location.
SurfaceCaptured: This command tells that the surface of interest has been captured.
TimeInState: This command indicates the time duration spent in a state.
TimeOut: This command indicates the time within which an event has to be conducted otherwise it is not executed.
Trim: This command is issued to increase speed of the vehicle.

Appendix B : Hybrid models in Teja

Sequential coordinator

Superclass: TejaComponent

Variables: NumWP, SurfaceThreshold, WPNum, Suspended, StartUTCTime, Suspendable, Idle.

Links: DevCmd (DeviceCmd), Actreq (ActionRequest), VehCmd (VehicleCmd), AutCmd (AutopilotCmd), Nav (NavState), Logs (Files), WPNav (WaypointNavigator), NonSeqController (TimedActions), Payload (PayloadDelivery), Components (ComponentList), Missionqueues (Queues), CurrOder (SeqOrder), DevCmdr (DeviceCommander), GPSFix (GPSFixer), Devstate (DeviceState), Waiter (Pause).

Functions:
ReadParams( ) is a function to read the parameters needed by the steer module to get executed successfully.
EndMission() is a function used to end a mission.
getMissionTime() is a function which returns the duration for which a mission is being executed till the current time.

Input: No input

Constructors:
DevCmd=p_devicecmd (initialzed to point to DeviceCmd)
ActReq=p_actionrequest (initialzed to point to ActionRequest)
VehCmd=p_vehiclecmd (initialzed to point to VehicleCmd)
AutCmd=p_autopilotcmd (initialzed to point to AutopilotCmd)
Nav=p_navstate (initialzed to point to NavState)
Logs=p_files (initialzed to point to Files)
WPNav=p_waypointnavigator (initialzed to point to Waypoint Navigator)
NonSeqController=p_timedactions (initialzed to point to TimedActions)
Payload=p_payloaddelivery (initialzed to point to PayloadDelivery)
Components=p_componentlist (initialzed to point to ComponentList)
MissionQueues=p_queues (initialzed to point to Queues)
DevCmdr=p_devicecommander (initialzed to point to DeviceCommander)
GPSFix=p_gpsfixer (initialzed to point to GPSFixer)
DevState=p_devicestate (initialzed to point to DeviceState)
Waiter=p_pause (initialzed to point to Pause)
WPNum=0 (initialzed to zero)

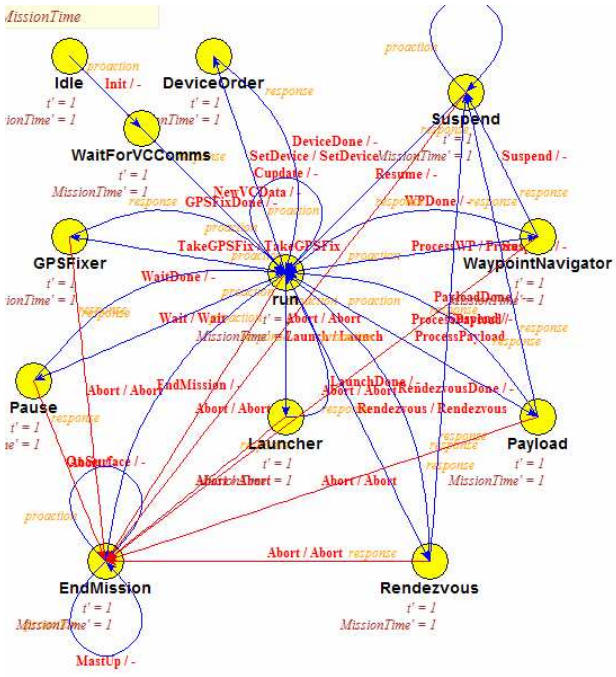


Figure 57: FSM for Sequential coordinator

Destructors: No destructors
Continuous states: t, MissionTime
Discrete states:
State1{ ***Idle***,t',MissionTime'}
State2{ ***DeviceOrder***, t',MissionTime'}
State3{ ***WaitForVCComms***, t',MissionTime'}
State4{ ***Suspend***, t',MissionTime'}
State5{ ***Pause***, t',MissionTime'}
State6{ ***GPSFixer***, t',MissionTime'}
State7{ ***run***, t',MissionTime'}
State8{ ***WaypointNavigator***, t',MissionTime'}
State9{ ***Launcher***, t',MissionTime'}
State10{ ***EndMission***, t',MissionTime'}
State11{ ***Rendezvous***, t',MissionTime'}
State12{***Payload***, t',MissionTime'}
Transitions:
Transition 1 {***Idle, WaitForVCComms,*** Init, t>=1, None, (CreateLogs(), ReadParams(), ReadOrderSpecs(), ReadMissionOrders() (and performs the actions as per need))}
Transition 2 {***WaitForVCComms, run, NewVCData***, True, (t=0, MissionTime=0, Suspendable=0, Idle=1), starts mission}
Transition 3 {***run, DeviceOrder, SetDevice***, True, (t=0, Suspendable=0, Idle=0), signal sent to device for execution}
Transition 4 {***DeviceOrder, run, DeviceDone***, True, (t=0, Suspendable=0, Idle=1), device command executed returns}
Transition 5 {***run, run, update***, t>=1, t=0, looks for new data or order availability}

Transition 6 {***Suspend, run, Resume***, True, (t=0, Suspended=0, Suspendable=0, Idle=1), resuming sequential execution}
Transition 7 {***Suspend, Suspend, Abort*****,** Suspended=True, Suspended=1}
Transition 8 {***WaypointNavigator, Suspend, Suspend***, True, (t=0, Suspendable=0, Idle=1), suspending waypoint}
Transition 9 {***Payload, Suspend, Suspend***, True, (t=0, Suspendable=0, Idle=1), suspending payload}
Transition 10 {***Rendezvous, Suspend, Suspend***, True, (t=0, Suspendable=0, Idle=1), suspending rendezvous}
Transition 11 {***Suspend, EndMission, Abort***, True, aborting from Suspend to endmission}
Transition 12 {***run, WaypointNavigator, ProcessWP***, True, (t=0, Suspendable=1, Idle=0), sending the datas to evaluate waypoint and perform actions accordingly}
Transition 13 {***WaypointNavigator, run, WPDone***, True, (t=0, Suspendable=0, Idle=1), mission to evaluate waypoint is executed}
Transition 14 {***run, Payload, ProcessPayload***, (CurrorderName=Payload), (t=0, Suspendable=1, Idle=0), evaluates waypoint to process payload}
Transition 15 {***Payload, run, PayloadDone***, True, (t=0, Suspendable=0, Idle=1), it states payload is done}
Transition 16 {***run, Launcher, Launch***, (CurrOrder=Launch), (Suspendable=0, Idle=0), launching vehicle }
Transition 17 {***Launcher, run, LaunchDone***, True, (t=0, Suspendable=0, Idle=1), launch has been done }
Transition 18 {***run, Pause, Wait***, (CurrOrder=Wait), (t=0, Suspendable=0, Idle=0), turning SSS and waiting }
Transition 19 {***Pause,run, WaitDone***, True, (t=0, Suspendable=0, Idle=1), sequential orders paused to wait to complete timed orders}
Transition 20 {***run, EndMission, EndMission***, (CurrOrder=EndMission and NonSeqController->Idle), end mission stopping prop and surfacing}
Transition 21 { ***EndMission, EndMission, OnSurface***, (Depth<=SurfaceThreshold and MastCmd!=Up), None, Comes on surface and mast is raised up}
Transition 22 { ***EndMission, EndMission, MastUp***, (MastState=Up), None, mast up and exiting mission}
Transition 23 {***Suspend, EndMission, Abort***, True, t=0, aborting from suspend }
Transition 24 {***WaypointNavigator, EndMission, Abort***, True, t=0, aborting from WaypointNavigator }
Transition 25 {***Payload, EndMission, Abort***, True, t=0, aborting from Payload}
Transition 26 {***Rendezvous, EndMission, Abort***, True, t=0, aborting from Rendezvous}
Transition 27 {***run, EndMission, Abort***, True, t=0, aborting from run}
Transition 28 {***Pause, EndMission, Abort***, True, t=0, aborting from Pause}
Transition 29 {***GPSFixer, EndMission, Abort***, True, t=0, aborting from GPSFixer}
Transition 30 {***Launcher, EndMission, Abort***, True, t=0, aborting from Launcher}

Timed Action (Timed Coordinator)

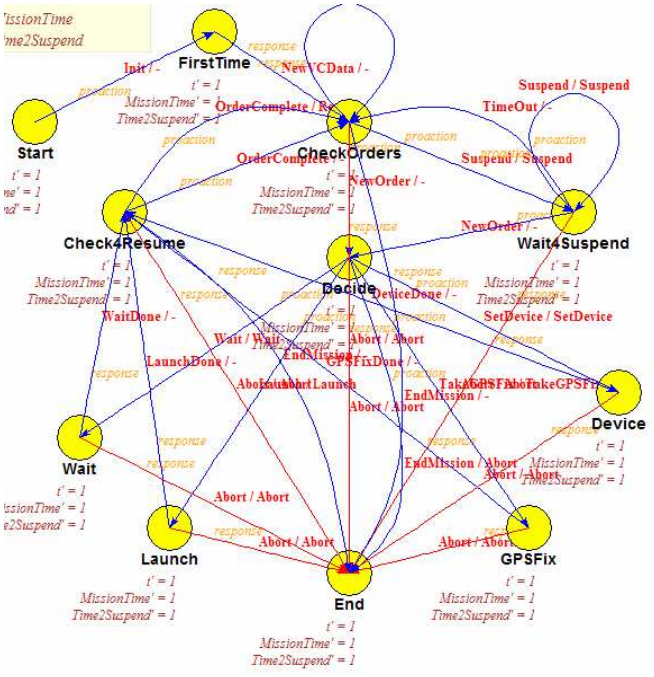


Figure 58: FSM of Timed coordinator

Superclass: TejaComponent

Variables: Token, TimedOrderTo, Suspend, Idle

Links: CurrTimeOrd (TimedOrder), Nav (NavState), NavMemory (NavState), SequenceController (Controller), Logs (Files), Components (ComponentList), MissionQueues (Queues), SeqOrdMemory (SeqOrder), CurrOrder (GPSOrder), GPSFix (GPSFixer), DevCmdr (DeviceCommander), Diver (Launcher), Waiter (Pause).

Functions:
ReadParams( ) is a function to read the parameters needed by the steer module to get executed successfully.
GetEarliestOrder() is used to retrieve the timed order with the earliest time requirement
CheckOrderTimes() is used to convert UTC times to mission times and check if > 0

Constructors:
MissionQueues=p_queues (initialzed to point to order Queue)
Nav=p_navstate (initialzed to point to NavState)
SeqController=p_controller (initialzed to point to Controller)
Logs=p_files (initialzed to point to Files)
Components=p_componentlist (initialzed to point to ComponentList)
GPSFix=p_gpsfixer (initialzed to point to GPSFixer)
DevCmdr=p_devicecommander (initialzed to point to DeviceCommander)
Diver=p_launcher (initialzed to point to Launcher)
Waiter=p_pause (initialzed to point to Pause)

NavMemory=NavState_new(teja_default_memory_space_id) (initialzed memory for NavState)

Desctructors: None

Continuous state: t, MissionTime

Discrete State:
State1{***Start***,t',MissionTime'}
State2{**FirstTime,**t',MissionTime'}
State3{***Decide***,t',MissionTime'}
State4{***CheckOrders***,t',MissionTime'}
State5{***GPSFix***,t',MissionTime'}
State6{***Device***,t',MissionTime'}
State7{***Wait***,t',MissionTime'}
State8{***Launch***,t',MissionTime'}
State9{**Wait4Suspend,** t', MissionTime'}
State10{**Check4Resume**, t', MissionTime'}
State11{**End**, t', MissionTime'}

Transitions:
Transition 1 {***Start,*** **FirstTime**, Init, t>=1, None, Checks for availability of data}
Transition 2 { **FirstTime*****,*** ***CheckOrders,*** NewVCData, MissionQueue>0, (MissionTime=0, Idle=0, t=0, Token=1), CheckOrderTimes and Loading Timed Order}
Transition 3 {***CheckOrders, End,*** Abort, True, None, Aborting}
Transition 4 {***Decide, End,*** Abort, True, None, Aborting Decide}
Transition 5 {***Check4Resume, CheckOrders,*** OrderComplete, (Token and TimedOrderQueue=0),(Idle=1, t=0, Token=1), No more Timed Orders so going to Idle state waiting for more tied orders}
Transition 6 {***Launch, End,*** Abort, True, None, Aborting Launch state}
Transition 7 {***Wait, End,*** Abort, True, None, Aborting Wait state}
Transition 8 {***Device, End,*** Abort, True, None, Aborting Device state}
Transition 9 {***GPSFix, End,*** Abort, True, None, Aborting GPSFix state}
Transition 10 {***Launch, Check4Resume,*** LaunchDone, ! (suspend and MissionQueue)=0, (Idle=1, t=0, Token=1), (teja_get_time(), TimedActions_get_MissionTime())}
Transition 11 {***Wait, Check4Resume,*** WaitDone, ! (suspend and MissionQueue)=0, (Idle=1, t=0, Token=1), (teja_get_time(),TimedActions_get_MissionTime())}
Transition 12 {***Device, Check4Resume,*** DeviceDone, ! (suspend and MissionQueue)=0, (Idle=0, t=0, Token=1), (teja_get_time(),TimedActions_get_MissionTime())}
Transition 13 {***GPSFix, Check4Resume,*** GPSFixDone, ! (suspend and MissionQueue)=0, (Idle=1, t=0, Token=0), (teja_get_time(),TimedActions_get_MissionTime())}

Transition 14 {***Wait4Suspend, Decide, NewOrder***, (TimedOrderQueue)>0, (t=0), (teja_get_time(),TimedActions_get_MissionTime(),CurrTimedOrd())}
Transition 15 {***CheckOrders, Wait4Suspend,*** Suspend, (TimedActions_get_MissionTime() >= CurrTimedOrd && Suspend), (Token=0,t=0), Store Current Nav State and Store Current sequential order }

Safeties (Safety Coordinator)

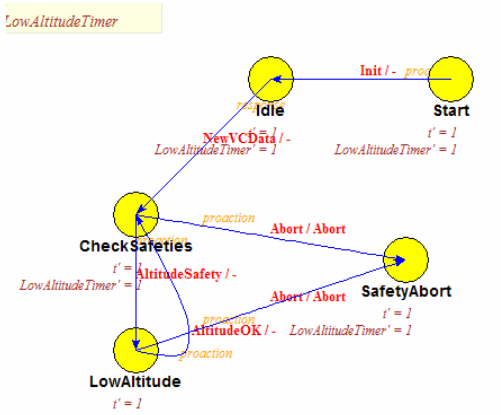


Figure 59: FSM of Safety coordinator

Superclass: TejaComponent

Variables: WaterDepthSafety, LowBatteryVoltage, MinimumAltitude, LowAltitudeTo.

Links: Bat(BatteryState), Nav(NavState), Logs(Files), Components(ComponentList).

Functions:
VoltageAbort() is a function which checks whether the average voltage is less than a threshold and accordingly returns a value.
WaterDepthAbort() is a function which checks whether the depth to which the vehicle can go to is safe or not and returns a value accordingly.
ReadParams() is a function to read the parameters needed by the safety module to get executed successfully.

Constructors:
Bat=p_batterystate; (initialzed to point to BatteryState)
Nav=p_navstate; (initialzed to point to NavState)
Logs=p_files; (initialzed to point to Files)
Components=p_componentlist; (initialzed to point to ComponentList)

Destructor: No destructors.

Continuous state: t, LowAltitudeTimer

Discrete States:

State 1 {***Start,*** t'=1, LowAltitudeTimer'=1}
State 2 {***Idle***, t'=1, LowAltitudeTimer'=1}
State 3 {***CheckSafeties***, t'=1, LowAltitudeTimer'=1}
State 4 {***SafetyAbort***, t'=1, LowAltitudeTimer'=1}
State 5 {***LowAltitude***, t'=1, LowAltitudeTimer'=1}
State 6 {***Error,*** t'=1, LowAltitudeTimer'=1}
State 7 {***Stop***, none}

Transitions:
Transition 1 {***Start, Idle***, Init, t>=1, none, ReadParams}
Transition 2 {***Idle, CheckSafeties***, NewVCData, True, t=0, (prints teja_get_time(), Safeties_get_t() in execlog and flushes execlog)}
Transition 3 {***CheckSafeties, LowAltitude, AltitudeSafety***, (Altitude< MinimumAltitude) , LowAltitudeTimer=0, None}
Transition 4 {***CheckSafeties, SafetyAbort, Abort***, (VoltageAbort or WaterDepthAbort), None, (VoltageAbort or WaterDepthAbort (print teja_get_time(),Safeties_get_t() into errorlog, flushes errorrlog))}
Transition 5 {***LowAltitude, SafetyAbort, Abort***, (Safeties_VoltageAbort() or Safeties_WaterDepthAbort() or Safeties_get_LowAltitudeTimer() >LowAltitudeTO), (Safeties_VoltageAbort or Safeties_WaterDepthAbort or Safeties_get_LowAltitudeTimer() > LowAltitudeTO) (print teja_get_time(), Safeties_get_t() to errorlog , flush errorlog finally) }
Transition 6 {***LowAltitude, Checksafeties, AltitudeOk***, Altitude>MinimumAltitude, None, None}
Transition 7 {***Error, Stop***, Error, True, None, None}

ReplayMission

The ReplayMission module is used to write a human readable commands file. It takes in the input commands and writes out them in formatted output file.

Superclass: TejaComponent

Variables: None

Links: Bat, Dev, Nav, Veh

Functions:
Quicklook() is a function which writes the commands in human readable form.

Constructors:
Bat=BatteryState_new(teja_default_memory_space_id,NUMBEROFBATTERYSWITCHES); (Initializes space to Bat of type BatteryState)
Dev=DeviceState_new(teja_default_memory_space_id); (Initializes space to Dev of type DeviceState)

Nav=NavState_new(teja_default_memory_space_id); (Initializes space to Nav of type NavState)
Veh=VehicleState_new(teja_default_memory_space_id); (Initializes space to Veh of type VehicleState)

Destructor: No destructors.

Continuous state: t

Discrete States:
State 1 {***Idle***, tâ€™=1}
State 2 {***Error***, tâ€™=1}
State 3 {***Stop***, None}

Transitions:
Transition 1 {***Idle, Idle***, Init, t>=1, None, (ReplayMission_Quicklook() print "Quicklook files created, ending Replay") }
Transition 2 {***Error, Stop***, Error, True, None, None}

GPSFixer

Superclass: TejaComponent

Variables: GoToSurfaceTo, RaiseMastTo, TakeMastTo, SurfaceThreshold, NumFailed, WPThresholdDistance.

Links: Nav (NavState), DevState (DeviceState), AutCmd(AutopilotCmd), DevCmd (DeviceCmd), VehCmd (VehicleCmd), ActReq (ActionRequest), Logs (Files), Components (ComponentList), NavMemory (NavState), Helm (Steering), GPSOrd (GPSOrder).

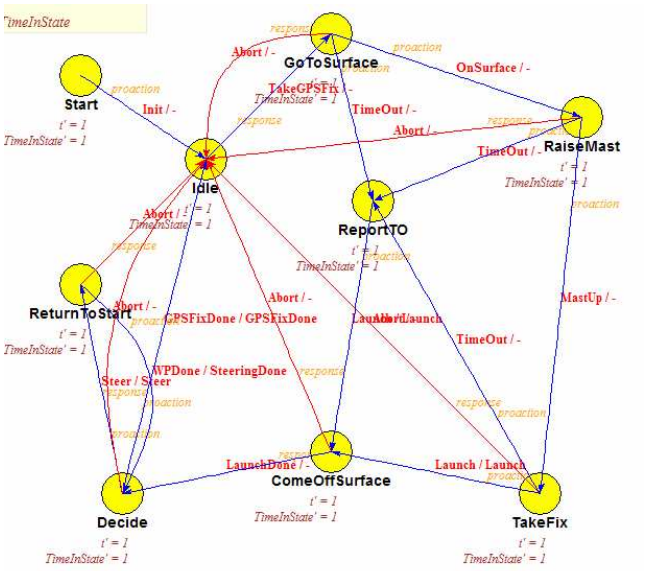


Figure 60: FSM for GPSFixer module

Functions:
ReadParams( ) is a function to read the parameters needed by the GPSFixer module to get executed successfully.

Constructor:
Nav=p_navstate; (initialzed to point to NavState)
DevState=p_devicestate; (initialzed to point to DeviceState)
AutCmd=p_autopilotcmd; (initialzed to point to AutopilotCmd)
DevCmd=p_devicecmd; (initialzed to point to DeviceCmd)
VehCmd=p_vehiclecmd; (initialzed to point to VehicleCmd)
ActReq=p_actionrequest; (initialzed to point to ActionRequest)
Logs=p_files; (initialzed to point to Files)
Components=p_componentlist; (initialzed to point to ComponentList)
Helm=p_steering; (initialzed to point to Steering)
NavMemory=NavState_new(teja_default_memory_space_id); (initialzed to point to NavState)
NumFailed=0; (initialzed to 0)

Destructor: No destructors.

Continuous state: t, TimeInState

Discrete States:
State 1 {***Start***, t'=1, TimeInState'=1}
State 2 {***Idle***, t'=1, TimeInState'=1}
State 3 {***GoToSurface***, t'=1, TimeInState'=1}
State 4 {***RaiseMast***, t'=1, TimeInState'=1}
State 5 {***ReportTo***, t'=1, TimeInState'=1}
State 6 {***TakeFix***, t'=1, TimeInState'=1}
State 7 {***ComeOffSurface***, t'=1, TimeInState'=1}
State 8 {***ReturnToStart***, t'=1, TimeInState'=1}
State 9 {***Decide***, t'=1, TimeInState'=1}
State 10 {***Error***, t'=1, TimeInState'=1}
State 11 {***Stop***, t'=1, TimeInState'=1}

Tranisitions:
Transition 1 {***Start, Idle***, Init, t>=1, none, ReadParams}
Transition 2 {***Idle, GoToSurface***, TakeGPSFix, True, (t=0, TimeInState=0), (Turning off prop and blowing tanks and performing operation according to VehCmd, DevCmd, AutCmd, ActReq)}
Transition 3 {***GoToSurface, Idle***, Abort, True, none, (Aborting and going to surface and print to errorlog (teja_get_time(),GPSFixer_get_t()), finally flush errorlog) }
Transition 4 {***RaiseMast***, ***Idle***, Abort, True, none, (Aborting raising mast and print to errorlog (teja_get_time(),GPSFixer_get_t()), finally flush errorlog) }

Transition 5 {***TakeFix, Idle***, Abort, True, none, (Aborting TakeFix and print to errorlog (teja_get_time(),GPSFixer_get_t()), finally flush errorlog) }
Transition 6 {***ComeOffSurface, Idle***, Abort, True, none, (Aborting ComeOffSurface and print to errorlog (teja_get_time(),GPSFixer_get_t()), finally flush errorlog) }
Transition 7 {***Decide, Idle***, Abort, True, TimeInState=0, (Aborting Decide and print to errorlog (teja_get_time(),GPSFixer_get_t()), finally flush errorlog) }
Transition 8 {***ReturnToStart, Idle***, Abort, True, none, (Aborting ReturnToStart and print to errorlog (teja_get_time(),GPSFixer_get_t()), finally flush errorlog) }
Transition 9 {***Decide, Idle***, GPSFixDone, GPSOrd=!ReturnToStart,TimeInState=0, (GPSFixDone and print to execlog (teja_get_time(),GPSFixer_get_t()), finally flush execlog) }
Transition 10 {***GoToSurface, ReportTo***, Timeout, TimeInState> GoToSurface, TimeInState=0, Print to file errorlog GoToSurface Timed Out teja_get_time(), GPSFixer_get_t()}
Transition 11 {***GoToSurface, RaiseMast,*** OnSurface, Depth <= SurfaceThreshold, TimeInState=0, Print to execlog teja_get_time(),GPSFixer_get_t() and finally fflush Execlog and excute VehCmd, DevCmd, ActReq)}
Transition 12 {***RaiseMast, ReportTo,*** TimeOut, TimeInState>= RaiseMastTo, TimeInState=0, Print to file errorlog RaiseMast Timed out teja_get_time(), GPSFixer_get_t()) and finally fflush errorlog}
Transition 13 {***RaiseMast, TakeFix***, MastUp, MastState=Up, TimeInState=0, execute VehCmd, ActReq and print to the file execlog GPSFixer - Mast up, waiting for GPS Fix, teja_get_time(),GPSFixer_get_t() and fflush execlog)}
Transition 14 {***TakeFix, ReportTo,*** TimeOut, (TimeInState>= TakeFixTo), TimeInState=0, Print TakeFix Timed Out, teja_get_time(),GPSFixer_get_t() in file errorlog finally fflush errorlog }
Transition 15 {***TakeFix, ComeOffsurface***, Launch, DevState->GPSFixState == DONE, TimeInState=0, Print Got GPS Fix, teja_get_time(),GPSFixer_get_t() into file execlog and finally fflush execlog) }
Transition 16 {***ReportTo, ComeOfsurface,*** Launch, True, None, (Print Time out, GPS Fix Done, teja_get_time(),GPSFixer_get_t() in file errorlog and finally fflush errorlog)}
Transition 17 {C***omeOfSurface, Decide***, LaunchDone, True, TimeInState=0, None}
Transition 18 {***Decide, ReturnToStart***, Steer, GPSOrd->ReturnToStart, TimeInState=0, Print ReturningToStartPt, teja_get_time(),GPSFixer_get_t() into file execlog and finally fflush execlog and executes VehCmd, AutCmd,DevCmd, ActReq commands) }
Transition 19 {***ReturnToStart, Decide***, WPDone, DistanceToPoint<= WPThresholdDistance, TimeInState=0, (Print at Start Point, teja_get_time(), GPSFixer_get_t() in file execlog, finally fflush execlog and GPSOrd->ReturnToStart = FALSE)}
Transition 20 {***Error, Stop***, Error, True, None, None}

Launcher

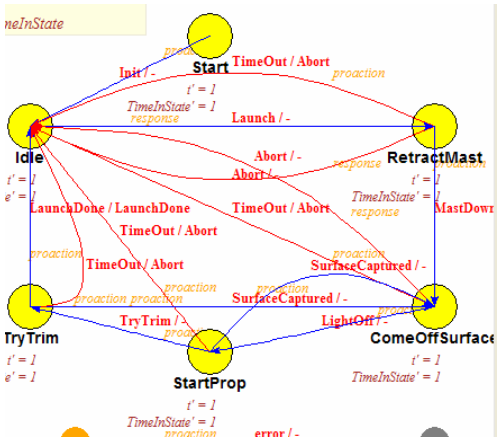


Figure 61: FSM of Launcher module

Superclass: TejaComponent

Variables: RetractMastTo, ComeOffSurfaceTo, LightOfDepth, FwdLaunchMast, AftlaunchMast.

Links: DevState (DeviceState), VehState(VehicleState), DevCmd(DeviceCmd), VehCmd(VehicleCmd), AutCmd(AutopilotCmd), ActReq(Actionreq), Logs(Files), Components(ComponentList), LaunchOrd(LaunchOrder), Nav(NavState).

Functions:
ReadParams( ) is a function to read the parameters needed by the Launcher module to get executed successfully.

Constructors: The data structures are initialized to point to their respective data structures.
DevState=p_devicestate; (initialzed to point to DeviceState)
VehState=p_vehiclestate; (initialzed to point to VehicleState)
DevCmd=p_devicecmd; (initialzed to point to DeviceCmd)
VehCmd=p_vehiclecmd; (initialzed to point to VehicleCmd)
AutCmd=p_autopilotcmd; (initialzed to point to AutopilotCmd)
ActReq=p_actionrequest; (initialzed to point to ActionReq)
Logs=p_files; (initialzed to point to Files)
Components=p_componentlist; (initialzed to point to ComponentList)
Nav=p_navstate; (initialzed to point to NavState)

Destructor: No destructors.

Continuous state: t, TimeInState.

Discrete States:
State 1 {***Start***, t'=1, TimeInState'=1}
State 2 {***Idle***, t'=1, TimeInState'=1}
State 3 {***RetractMast***, t'=1, TimeInState'=1}

State 4 {***ComOffSurface*****,** t'=1, TimeInState'=1}
State 5 {***StartProp***, t'=1, TimeInState'=1}
State 6 {***TryTrim***, t'=1, TimeInState'=1}
State 7 {***Error***, t'=1, TimeInState'=1}
State 8 {***Stop***, t'=1, TimeInState'=1}

Transitions:
Transition 1 {***Start, Idle***, Init, t>=1, none, ReadParams}
Transition 2 {***RetractMast, Idle***, TimeOut, TimeInState >=RetractMastTo, none, (Print in file errorlog Launch - Retract Mast Timed Out - Aborting Mission, teja_get_time(), Launcher_get_t() and finally fflush errorlog)}
Transition 3 {***Idle, RetractMast,*** Launch, True, (t=0, TimeInstate=0), (Print in file execlog Launch - Retracting Mast, teja_get_time(), Launcher_get_t() and finally fflush execlog and execute commands by VehCmd, DevCmd, DevState, ActReq)}
Transition 4 {***ComeOffSurface, Idle,*** Abort, True, (Print in file errorlog Launch - Aborting ComeOffSurface on signal, teja_get_time(), Launcher_get_t() and finally fflush errorlog)}
Transition 5 {***RetractMast, Idle,*** Abort, True, (Print in file errorlog Launch - Aborting RetractMast on signal, teja_get_time(), Launcher_get_t() and finally fflush errorlog)}
Transition 6 {***ComeOffSurface, Idle,*** TimeOut, TimeInState>=ComeOffSurfaceTo, (Print in file errorlog Launch - ComeOffSurface Timed Out - Aborting Mission, teja_get_time(), Launcher_get_t() fflush errorlog)}
Transition 7 {***TryTrim, Idle,*** LaunchDone, (Speed > 1.5 && Launcher_get_TimeInState() > 60.0), None, (Print in file execlog Launch - LaunchDone, teja_get_time(),Launcher_get_t() and finally fflush execlog)}
Transition 8 {**RetractMast, ComeOffSurface**, MastDown, True, TimeInState=0, (Print to file execlog Launch - Mast Down, coming off surface, teja_get_time(), Launcher_get_t() and finally fflush execlog and (Altitude >20 LightOffdepth= 10 or Altitude >10 LightOffDepth=5 or LightOffDepth=5) and executes VehCmd, DevCmd and ActReq)}
Transition 9 {***ComeOffSurface, StartProp,*** LightOff, Depth >= LightOffDepth, (Print in file execlog Launch - Lighting-off prop, teja_get_time(), Launcher_get_t() and finally fflush execlog and execute commands from VehCmd, AutCmd, ActReq) }
Transition 10 {***StartProp, ComeOffSurface,*** SurfaceCaptured, Depth<1.5, TimeInState=0, (Print to file execlog Launch - Surface Captured - trying to come off surface, teja_get_time(), Launcher_get_t() and finally fflush execlog and (Altitude >20 LightOffdepth= 10 or Altitude >10 LightOffDepth=5 or LightOffDepth=5) and executes VehCmd, DevCmd and ActReq)}
Transition 11 {***TryTrim, ComeOffSurface***, SurfaceCaptured, Depth<1.5, TimeInState=0, (Print to file execlog Launch - Surface Captured - trying to come off surface, teja_get_time(), Launcher_get_t() and finally fflush execlog and (Altitude >20 LightOffdepth= 10 or Altitude >10 LightOffDepth=5 or LightOffDepth=5) and executes VehCmd, DevCmd and ActReq)}
Transition 12 {***StartProp, TryTrim,*** TryTrim, Speed>1, TimeInState=0, (Print in file execlog Launch - Trying VBS Trim to get fin authority, teja_get_time(),

Launcher_get_t() and finally fflush execlog and execute commands from VehCmd, DevCmd, ActReq)}
Transition 13 {***Error, Stop***, Error, True, None, None}

WayPointnavigator

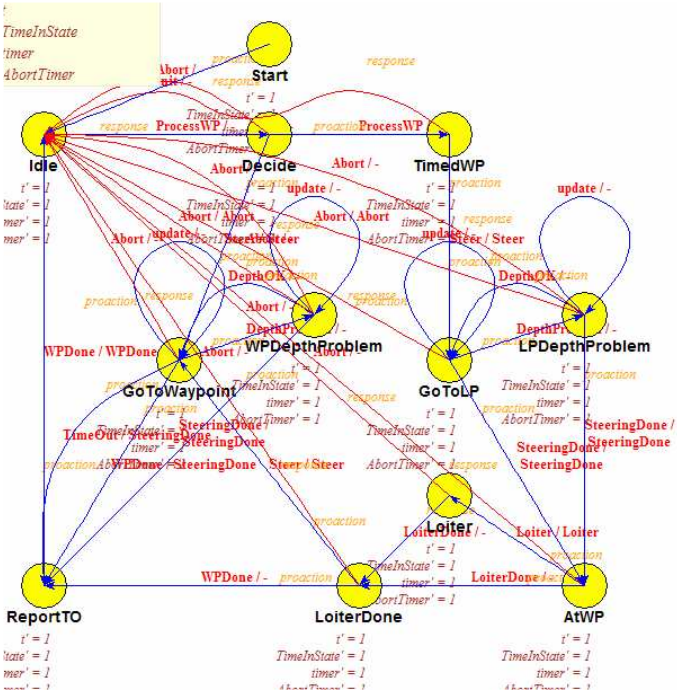


Figure 62: FSM of Waypointnavigator

Superclass: TejaComponent

Variables: WayPointPctOverrun, WayPointTo, ThresholdDistance, MinSpeed, MaxSpeed, TimeToWayPoint, DistanceToWayPoint, LoiterDistance, LoiterAwayDistance, Requestor.

Links: Nav (NavState), ToWP (WayPoints), AutCmd (AutopilotCmd), ActReq (ActionReq), DevCmd (DeviceCmd), VehCmd (VehicleCmd), FromWP (Wayoints), Logs (Files), Helm (Steering), MC (Controller), Vagabond (Loiter), Components (ComponentList)

Functions:
ReadParams( ) is a function to read the parameters needed by the WayPointNavigator module to get executed successfully.

Constructors: The data structures are initialized to point to their respective data structures.

Nav=p_navstate; (initialzed to point to NavState)

AutCmd=p_autopilotcmd; (initialzed to point to AutopilotCmd)
ActReq=p_actionrequest; (initialzed to point to ActionReq)
DevCmd=p_devicecmd; (initialzed to point to DevCmd)
VehCmd=p_vehiclecmd; (initialzed to point to VehicleCmd)
Helm=p_steering; (initialzed to point to Steering)
MC=p_controller; (initialzed to point to Controller)
Logs=p_files; (initialzed to point to Files)
Vagabond=p_loiter; (initialzed to point to Loiter)
Components=p_componentlist; (initialzed to point to ComponetList)
ToWP=Waypoints_new(teja_default_memory_space_id); (initialzed to point to WayPoint)
FromWP=Waypoints_new(teja_default_memory_space_id); (initialzed to point to WayPoint)

Destructor: No destructors.

Continuous state: t, TimeInState, timer

Discrete States:
State 1 {***Start***, t'=1, TimeInState'=1, timer'=1}
State 2 {***Idle***, t'=1, TimeInState'=1, timer'=1}
State 3 {***Decide***, t'=1, TimeInState'=1, timer'=1}
State 4 {***TimedWP***, t'=1, TimeInState'=1, timer'=1}
State 5 {***GoToWayPoint***, t'=1, TimeInState'=1, timer'=1}
State 6 {***GoToWP***, t'=1, TimeInState'=1, timer'=1}
State 7 {***Loiter***, t'=1, TimeInState'=1, timer'=1}
State 8 {***AtWP***, t'=1, TimeInState'=1, timer'=1}
State 9 {***LoiterDone***, t'=1, TimeInState'=1, timer'=1}
State 10 { ***ReportTo***, t'=1, TimeInState'=1, timer'=1}
State 11 {***Error,*** t'=1, TimeInState'=1, timer'=1}
State 12 {***Stop***, none}

Transitions:
Transition 1 {***Start, Idle***, Init, t>=1, none, (ReadParams, LoiterAwayDistance < 500.0) { LoiterAwayDistance=500.0, Print in file errorlog WaypointNavigator - LoiterAwayDistance too small, resetting to 500 m, teja_get_time(), WaypointNavigator_get_t() and finally fflush errorlog)}
Transition 2 {***Idle, Decide***, ProcessWP, True, t=0, Calculate the distance to cover}
Transition 3 {***ReportTo, Idle,*** WPDone, True, TimeInState=0, None (Output)}
Transition 4 {***Decide, Idle***, Abort, True, None, (Print in file errorlog WaypointNavigator - Aborting on signal, teja_get_time(), WaypointNavigator_get_t() finally fflush errorlog)}
Transition 5 {***TimedWP, Idle***, Abort, True, None, (Print in file errorlog WaypointNavigator - Aborting on signal, teja_get_time(), WaypointNavigator_get_t() finally fflush errorlog)}

Transition 6 {***GoToWayPoint, Idle***, Abort, True, None, (Print in file errorlog WaypointNavigator - Aborting on signal, teja_get_time(), WaypointNavigator_get_t() finally fflush errorlog)}
Transition 7 {***GoToWP, Idle***, Abort, True, None, (Print in file errorlog WaypointNavigator - Aborting on signal, teja_get_time(), WaypointNavigator_get_t() finally fflush errorlog)}
Transition 8 {***AtWP, Idle***, Abort, True, None, (Print in file errorlog WaypointNavigator - Aborting on signal, teja_get_time(), WaypointNavigator_get_t() finally fflush errorlog)}
Transition 9 {***Loiter, Idle***, Abort, True, None, (Print in file errorlog WaypointNavigator - Aborting on signal, teja_get_time(), WaypointNavigator_get_t() finally fflush errorlog)}
Transition 10 {***LoiterDone, Idle***, Abort, True, None, (Print in file errorlog WaypointNavigator - Aborting on signal, teja_get_time(), WaypointNavigator_get_t() finally fflush errorlog)}
Transition 11 {***Decide, TimedWP***, ProcessWP, (ToWP->Timed && TimeToWaypoint > DistanceToWaypoint/MinSpeed), None, Loiters to desired point with desired Loiter type}
Transition 12 {***Decide, GoToWayPoint***, Steer, (!ToWP->Timed || TimeToWaypoint <= DistanceToWaypoint/MinSpeed), (TimeInState=0, timer=0), Goes to desired latitude and longitude with desired steer mode}
Transition 13 {***TimedWP, GoToWP,*** True, timer=0, (Goes to desired latitude and longitude with desired SteerMode, HeadingMode, DepthMode)}
Transition 14 {***GoToWP, GoToWP,*** Update, timer>=1, timer=0, Calculate the distance to desired waypoint}
Transition 15 {***GoToWP, AtWP,*** Abort, (Helm->DistanceToPoint <= LoiterDistance) || (TimeToWaypoint-WaypointNavigator_get_t() <= DistanceToWaypoint/MinSpeed), None, Calculates by how much the vehicle loiters away from the desired waypoint }
Transition 16 {***AtWP, Loiter***, Loiter, ToWP->LoiterDuration>0, None, (Print in file execlog WaypointNavigator - Going to loiter, teja_get_time(), WaypointNavigator_get_t() fflush execlog) }
Transition 17 {***AtWP, LoiterDone,*** LoiterDone, ToWP->LoiterDuration<=0, None, (Print in file errorlog WaypointNavigator - No time to loiter , teja_get_time(), WaypointNavigator_get_t() fflush errorlog)}
Transition 18 {***Loiter, LoiterDone,*** LoiterDone, True, None, None}
Transition 19 {***LoiterDone,GoToWayPoint***, Steer, (!ToWP->LoiterAtWP && TimeToWaypoint >= WaypointNavigator_get_t()), (TimeInState=0, timer=0), Calculate the distance to desired latitude and longitude with desired speedmode}
Transition 20 {***LoiterDone, ReportTo,*** WPDone, (ToWP->LoiterAtWP || (TimeToWaypoint <= WaypointNavigator_get_t())), None, None}
Transition 21 {***GoToWayPoint, GoToWayPoint,*** Update, timer>=2, timer=0, Calculates the time to go to the desired location with the desired speedmode}
Transition 22 { ***GoToWayPoint, ReportTo***, WPDone, ((!ToWP->Timed && Helm->DistanceToPoint <= ThresholdDistance) || (ToWP->Timed && Helm->DistanceToPoint <= 5.0)), TimeInState=0, (Print in file execlog WaypointNavigator - WP Done, teja_get_time(), WaypointNavigator_get_t() and finally fflush execlog)}

Transition 23 { ***GoToWayPoint, ReportTo***, TimeOut, TimeInState>=WayPointTo, (Print in file errorlog WaypointNavigator - Waypoint Timed Out, teja_get_time(), WaypointNavigator_get_t() and finally fflush errorlog)}
Transition 24 {***ReportTo, Idle,*** WPDone, True, TimeInstate=0, None}
Transition 25 {***Error, Stop***, Error, True, None, None}

Rendezvous

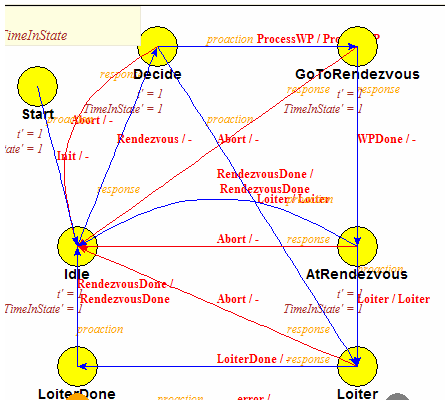


Figure 63: FSM of Rendezvous module

Superclass: TejaComponent

Variables: WPThreshDistMemory

Links: ActReq, Logs, Vagabond, WPNav, DevCmd, VehCmd, Components

Functions: None

Constructors:
ActReq=p_actionrequest; (initialzed to point to Actionreq)
Logs=p_files; (initialzed to point to Files)
Vagabond=p_loiter; (initialzed to point to Loiter)
WPNav=p_waypointnavigator; (initialzed to point to WayPointNavigator)
DevCmd=p_devicecmd; (initialzed to point to DeviceCmd)
VehCmd=p_vehiclecmd; (initialzed to point to VehicleCmd)
Components=p_componentlist; (initialzed to point to ComponentList)

Destructor: No destructors.

Continuous state: t, TimeInState
Discrete States:
State 1 {***Start***, t'=1, TimeInState'=1}
State 2 {***Idle***, t'=1, TimeInState'=1}
State 3 {***LoiterDone***, t'=1, TimeInState'=1}
State 4 {***Decide***, t'=1, TimeInState'=1}

State 5 {***GoToRendezvous***, t'=1, TimeInState'=1}
State 6 {***AtRendouzvous***, t'=1, TimeInState'=1}
State 7 {***Loiter***, t'=1, TimeInState'=1}
State 8 {***Error***, t'=1, TimeInState'=1}
State 9 {***Stop***, none}

Transitions:
Transition 1 {***Start, Idle***, Init, t>=1, None, None}
Transition 2 {***Decide, Idle***, Abort, True, None, (Print in file errorlog Rendezvous - Aborting, teja_get_time(),Rendezvous_get_t() and finally fflush errorlog)}
Transition 3 {***Idle, Decide,*** Rendezvous, True, (t=0, TimeInState=0), Calculates the Way point threshold distance depending upon the loiter type used for movement}
Transition 4 {***GoToRendezvous, Idle,*** Abort, True, None, (WPNav->ThresholdDistance=WPThreshDistMemory, Print in file errorlog,Rendezvous - Aborting, teja_get_time(), Rendezvous_get_t(), and finally fflush errorlog)}
Transition 5 {***AtRendezvous, Idle***, RendezvousDone, (WPNav->ToWP->LoiterType==NONE || WPNav->ToWP->LoiterDuration==0.0), None, (WPNav->ThresholdDistance= WPThreshDistMemory, Print in file execlog Rendezvous - No loiter specified, leaving rendezvous, teja_get_time(), Rendezvous_get_t() and finally fflush execlog)}
Transition 6 {***AtRendezvous, Idle,*** Abort, True, None, (WPNav->ThresholdDistance= WPThreshDistMemory, Print to file errorlog: Rendezvous â€“ Aborting, teja_get_time(), Rendezvous_get_t() and finally fflush errorlog)}
Transition 7 {***Loiter, Idle,*** Abort, True, None, (WPNav->ThresholdDistance= WPThreshDistMemory, Print to file errorlog: Rendezvous â€“ Aborting, teja_get_time(), Rendezvous_get_t() and finally fflush errorlog)}
Transition 8 {***LoiterDone, Idle,*** RendezvousDone, True, TimeInState=0, (WPNav->ThresholdDistance= WPThreshDistMemory, Print to file execlog: Rendezvous - Leaving rendezvous, teja_get_time(), Rendezvous_get_t() and finally fflush execlog)}
Transition 9 {***Decide, GoToRendezvous***, ProcessWP, True, None, executes commands given by VehCmd, DevCmd, ActReq}
Transition 10 {***Decide, Loiter***, Loiter, (WPNav->ToWP->Latitude==0 && WPNav->ToWP->Longitude==0 && WPNav->ToWP->Depth==0 && WPNav->ToWP->Speed==0), TimeInState=0, Updates various loiter variables and goes of to loiter when specified rendezvous point is not given}
Transition 11 {***GoToRendezvous, AtRendezvous***, WPDone, True, None, None}
Transition 12 {***AtRendezvous, Loiter***, Loiter, True, TimeInState=0, (Print in file excelog Rendezvous - Going to Loiter, teja_get_time(),Rendezvous_get_t(), and finally fflush execlog and loiter to different locations)}
Transition 13 {***Loiter, LoiterDone***, LoiterDone, True, None, None}
Transition 14 {***Error, Stop***, Error, True, None, None}

DeviceCommander

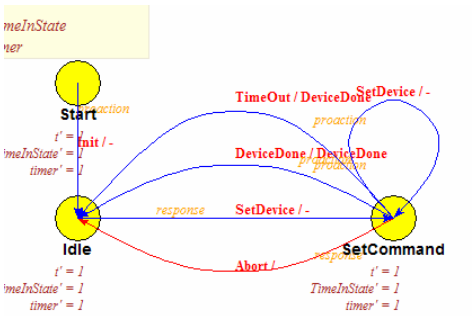


Figure 64: FSM for DeviceCommander

Superclass: TejaComponent
Variables: CmdSet, GoSurfaceTo, RaiseMastTo, SurfaceThreshold, ComeOffSurfaceTo, RetractMastTo, SetSwitchTo

Links: Components (ComponentList), Logs (Files), DevOrd (DeviceOrd),DevCmd (DeviceCmd), DevState (DeviceState), ActReq (ActionReq), VehCmd (VehicleCmd), Nav (NavState), InitDevState (DeviceState), AutCmd (AutopilotCmd), InitNav (NavState)

Functions:
SetDeviceCmd( ) is a function to set the device command specified in the current device command order.
DeviceCmdDone( ) is a function to check if the current device command is completed
CheckDeviceCmd( ) is a function which checks if it is safe to apply current device command
ReadParams( ) is a function to read the parameters needed by the DeviceCommander module to get executed successfully.
TimeOut( ) is a function to check if a time out for the current device has occurred

Constructors:
Components=p_componentlist; (initialzed to point to ComponentList)
Logs=p_files; (initialzed to point to Files)
DevCmd=p_devicecmd; (initialzed to point to DeviceCmd)
DevState=p_devicestate; (initialzed to point to DeviceState)
VehCmd=p_vehiclecmd; (initialzed to point to VehicleCmd)
ActReq=p_actionrequest; (initialzed to point to ActionReq)
Nav=p_navstate; (initialzed to point to NavState)
AutCmd=p_autopilotcmd; (initialzed to point to AutopilotCmd)
SetSwitchTO=5.0;
InitDevState=DeviceState_new(teja_default_memory_space_id); (initializing memory space for DeviceState)
InitNav=NavState_new(teja_default_memory_space_id); (initializing memory space for NavState)

Destructor: No destructors.

Continuous state: t, TimeInState, timer

Discrete States:
State 1 {***Start***, t'=1, TimeInState'=1, timer'=1}
State 2 {***Idle***, t'=1, TimeInState'=1, timer'=1}
State 3 {***SetCommand***, t'=1, TimeInState'=1, timer'=1}
State 4 {***Error***, t'=1, timer'=1}
State 5 {***Stop***, none}

Transitions:
Transition 1 {***Start, Idle***, Init, t>=1, none, ReadParams}
Transition 2 {***Idle, SetCommand***, SetDevice, True, t=0, TimeInState=0, timer=0, (Saves initial device states,saves initial nav state, (CmdSet=True, Device Commander - Setting Device Commands) (CmdSet= False, Device Commander - Waiting to set Device Commands))}
Transition 3 {***SetCommand, Idle***, DeviceDone, (DeviceCommander_DeviceCmdDone() && CmdSet), TimeInState=0, (Print Commander - Device command done, teja_get_time(),DeviceCommander_get_t())}
Transition 4 {***SetCommand, Idle***, TimeOut, DeviceCommander_TimeOut(this), None, None)
Transition 5 {***SetCommand, Idle***, Abort, True, None,(Print to file errorlog DeviceCommander - Aborting set device command, teja_get_time(), DeviceCommander_get_t() and finally fflush errorlog)
Transition 6 {***SetCommand, SetCommand,*** SetDevice, (DeviceCommander_get_timer() >=1 && !CmdSet), timer=0, (DeviceCommander_CheckDeviceCmd DeviceCommander_SetDeviceCmd() CmdSet=TRUE, Print to file execlog Device Commander - Setting Device Commands, teja_get_time(), DeviceCommander_get_t())}
Transition 7 {***Error, Stop***, Error, True, None, None}

PayloadDelivery

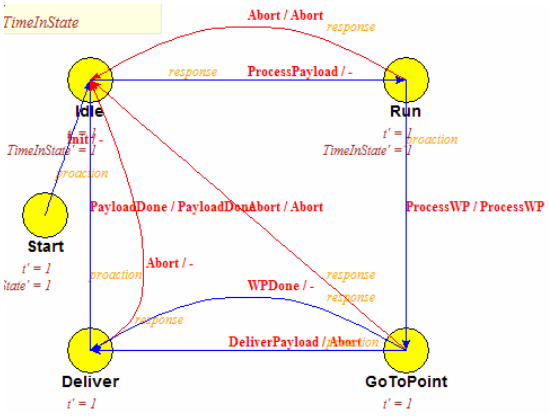


Figure 65: FSM of Payload module

Superclass: TejaComponent

Variables: DeliveryDelay, PayloadDescription, WPDistThreshMemory

Links: Logs, WPNav, VehCmd, ActReq, DevCmd, AutCmd, Components

Functions:
ReadParams( ) is a function to read the parameters needed by the PayloadDelivery module to get executed successfully.

Constructors:
AutCmd=p_autopilotcmd; (initialzed to point to AutopilotCmd)
WPNav=p_waypointnavigator; (initialzed to point to WayPointNavigator)
VehCmd=p_vehiclecmd; (initialzed to point to VehicleCmd)
ActReq=p_actionrequest; (initialzed to point to ActionReq)
DevCmd=p_devicecommand; (initialzed to point to DeviceCommand)
Logs=p_files; (initialzed to point to Files)
Components=p_componentlist; (initialzed to point to ComoponentList)

Destructor: No destructors.

Continuous state: t, TimeInState

Discrete states:
State 1 {***Start***, t'=1, TimeInState'=1}
State 2 {***Idle***, t'=1, TimeInState'=1}
State 3 {***Run***, t'=1, TimeInState'=1}
State 4 {***Deliver***, t'=1, TimeInState'=1}
State 5 {***GoToPoint***, t'=1, TimeInState'=1}
State 6 {***Error,*** t'=1, timer'=1}
State 7 {***Stop***, none}

Transitions:
Transition 1 {***Start, Idle***, Init, t>=1, none, ReadParams}
Transition 2 {***Deliver, Idle,*** PayloadDone, TimeInState>=DeliveryDelay, TimeInState=0, (WPNav->ThresholdDistance=WPDistThreshMemory and executes commands DevCmd, Acteq and prints to file execlog Payload - Payload Delivery Done, teja_get_time(), PayloadDelivery_get_t() and finally fflush execlog)}
Transition 3 {***Idle, Run***, ProcessPayload, True, (t=0, TimeInState=0), (WPDistThreshMemory=WPNav->ThresholdDistance)}
Transition 4 {***Run, Idle,*** True, None, (Print to file errorlog PayloadDelivery - Aborting, teja_get_time(), PayloadDelivery_get_t(), and finally fflush errorlog)}
Transition 5 {***GoToPoint, Idle***, Abort, True, None, (print to errorlog PayloadDelivery - Aborting, teja_get_time(), PayloadDelivery_get_t() and finally fflush errorlog)}
Transition 6 {***Deliver, Idle***, Abort, True, (print to errorlog PayloadDelivery - Aborting, teja_get_time(), PayloadDelivery_get_t() and finally fflush errorlog)}
Transition 7 {***Run, GoToPoint***, ProcessWP, True, None, (WPNav->ThresholdDistance=5.0 print to file execlog Payload - Proceeding to Payload Delivery

Point, teja_get_time(), PayloadDelivery_get_t() and finally fflush execlog and commands in VehCmd, DevCmd, ActReq are operated)}
Transition 8 {***GoToPoint, Deliver***, DeliverPayload, (WPNav->TimeToWaypoint <= DeliveryDelay), TimeInState=0, (execute commands VehCmd, DevCmd, ActReq, PayloadDescription==PORT print to file execlog Payload - Deliver port payload, teja_get_time(),PayloadDelivery_get_t() or PayloadDescription==STBD print to file execlog Payload - Deliver port payload, teja_get_time(),PayloadDelivery_get_t() or PayloadDescription==BOTH print to file execlog Payload - Deliver port payload, teja_get_time(),PayloadDelivery_get_t() and finally fflush execlog)}
Transition 9 {***GoToPoint, Deliver***, WPDone, True, TimeInState=0, (execute commands VehCmd, DevCmd, ActReq, PayloadDescription==PORT print to file execlog Payload - Deliver port payload, teja_get_time(),PayloadDelivery_get_t() or PayloadDescription==STBD print to file execlog Payload - Deliver port payload, teja_get_time(),PayloadDelivery_get_t() or PayloadDescription==BOTH print to file execlog Payload - Deliver port payload, teja_get_time(),PayloadDelivery_get_t() and finally fflush execlog)}
Transition 10 {***Error, Stop***, Error, True, None, None}

Loiter

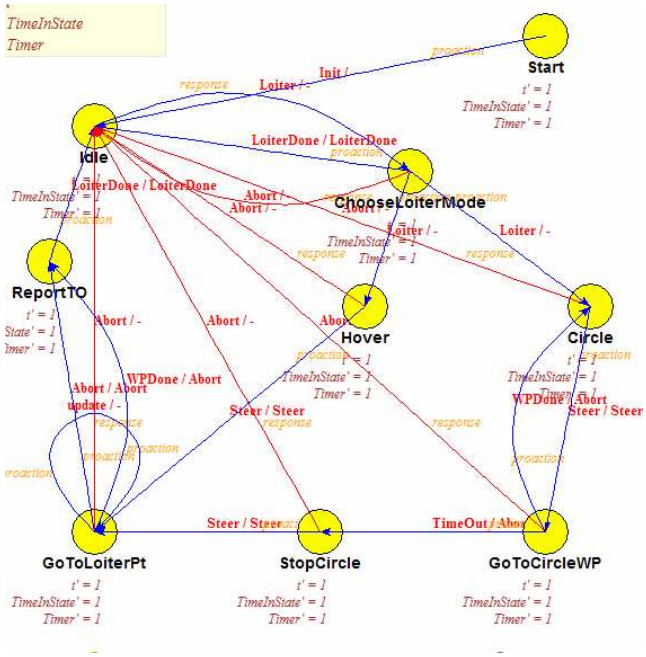


Figure 66: FSM of Loiter module

Superclass: TejaComponent

Variables: LoiterTo, Radius, LoiterLat, LoiterLon, LoiterDepth, LoiterSpeed, CircleLats, CircleLons, NumPoints, CurrentWP, LoiterSpeedMode, MinSpeed, MaxSpeed, Requestor

Links: Logs, Nav, WPNav, ActReq, VehCmd, AutCmd, DevCmd, Helm, Components

Functions:
ReadParams( ) is a function to read the parameters needed by the Loiter module to get executed successfully.
PlotCircle( ) is the function which finds the latitudes and longitudes of points that define a circle about the Loiter Point.
TimeToLoiterPoint( ) is a function that calculates time to loiter point.

Constructors:
Logs=p_files; (initialzed to point to Files)
Nav=p_navstate; (initialzed to point to NavState)
WPNav=p_waypointnavigator; (initialized to point to WayPointNavigator)
ActReq=p_actionrequest; (initialzed to point to ActionRequest)
VehCmd=p_vehiclecmd; (initialzed to point to VehicleCmd)
AutCmd=p_autopilotcmd; (initialzed to point to AutopilotCmd)
DevCmd=p_devicecmd; (initialzed to point to DeviceCmd)
Helm=p_steering; (initialzed to point to Steering)
Components=p_componentlist; (initialzed to point to ComponentList)
LoiterSpeedMode=OPENLOOP;
CurrentWP=0;

Destructor: No destructors.

Continuous state: t, TimeInState, timer

Discrete states:
State 1 {***Start***, t'=1, TimeInState'=1, timer'=1}
State 2 {***Idle***, t'=1, TimeInState'=1, timer'=1}
State 3 {***ReportTo***, t'=1, TimeInState'=1, timer'=1}
State 4 {***CooseLoiterMode***, t'=1, TimeInState'=1, timer'=1}
State 5 {***Hover***, t'=1, TimeInState'=1, timer'=1}
State 6 {***Circle***, t'=1, TimeInState'=1, timer'=1}
State 7 {***GoToCircleWP***, t'=1, TimeInState'=1, timer'=1}
State 8 {***StopCircle***, t'=1, TimeInState'=1, timer'=1}
State 9 {***GoToLoiterPt***, t'=1, TimeInState'=1, timer'=1}
State 10 {***Error,*** t'=1, TimeInState'=1, timer'=1}
State 11 {***Stop***, none}

Transitions:
Transition 1 {***Start, Idle***, Init, t>=1, none, ReadParams}
Transition 2 {***Idle***,C***hooseLoiterMode,*** Loiter, True, t=0, None}
Transition 3 {***ChooseLoiteMode, Idle***, LoiterMode, (NumPoints<2 && LoiterTO==0) || WPNav->ToWP->LoiterType==NONE), None, (Print to file execlog Loiter - No loiter required....Ending Loiter, teja_get_time(), Loiter_get_t() and finally fflush execlog)}
Transition 4 {***ChooseLoiteMode, Idle***, Abort, True, None, (Print to file execlog Loiter - Aborting from Loiter on abort signal, teja_get_time(), Loiter_get_t() and finally

fflush execlog)}
Transition 5 {***Hover, Idle***, Abort, True, (t=0, TimeInState=0), (Print to file execlog Loiter - Aborting from Loiter on abort signal, teja_get_time(), Loiter_get_t() and finally fflush execlog)}
Transition 6 {***Circle, Idle***, Abort, True, (t=0, TimeInState=0), (Print to file execlog Loiter - Aborting from Loiter on abort signal, teja_get_time(), Loiter_get_t() and finally fflush execlog)}
Transition 7 {***GoToCircleWP, Idle***, Abort, True, None, (Print to file execlog Loiter - Aborting from Loiter on abort signal, teja_get_time(), Loiter_get_t() and finally fflush execlog)}
Transition 8 {***StopCircle, Idle***, Abort, True, (t=0, TimeInState=0), (Print to file execlog Loiter - Aborting from Loiter on abort signal, teja_get_time(), Loiter_get_t() and finally fflush execlog)}
Transition 9 {***GoToLoiterPt, Idle***, Abort, True, (t=0, TimeInState=0), (Print to file execlog Loiter - Aborting from Loiter on abort signal, teja_get_time(), Loiter_get_t() and finally fflush execlog)}
Transition 10 {***ReportTo, Idle***, LoiterDone, True, None, None}
Transition 11 {***ChooseLoiterMode, Hover***, Loiter, (NumPoints<2 && LoiterTO>0) || WPNav->ToWP->LoiterType==HOVER), TimeInState=0, (Print to file execlog Loiter - Setting VBS in Hover mode, teja_get_time(),Loiter_get_t() and finally fflush execlog and execute commands in VehCmd, AutCmd, DevCmd, ActReq)}
Transition 12 {***ChooseLoiterMode, Circle***, Loiter, (NumPoints>=2 && LoiterTO>0 && WPNav->ToWP->LoiterType==CIRCLE), TimeInState=0, (Print to file execlog Loiter - Computing Loiter Circle and going to first point, teja_get_time(),Loiter_get_t() and finally fflush execlog and execute commands in VehCmd, AutCmd, ActReq and Loiter_PlotCircle)
Transition 13 {***Circle, GoToCircleWP***, Steer, NumPoints>=2, (WPNav->ToWP->UseSSS) (Print to execlog Loiter - Turning On SSS, teja_get_time(),Loiter_get_t() fflush execlog, DevCmd->SSSCmd=ON) or (DevCmd->SSSCmd=OFF DevCmd->VBSCmd=TRIM, DevCmd->VBSDepthCmd=WPNav->ToWP->Depth)}
Transition 14 {***GoToCircleWP, Circle***, WPDone, Helm->DistanceToPoint<=20.0, None, Loiter - Processing loiter waypoint}
Transition 15 {***Hover, GoToLoiterpt***, (TimeInState>= LoiterTO-Loiter_TimeToLoiterPt() â€“ 20), Timer=0, (Loiter - Turning On SSS or Loiter â€“ returningToLoiterPt)}
Transition 16 {***GoToCircleWP, StopCircle***, TimeOut, (TimeInState >= LoiterTO-Loiter_TimeToLoiterPt(), None, None}
Transition 17 {***StopCircle, GoToLoiterpt***, Steer, True, None, Loiter â€“ Returning

Appendix C: OpenGL Code for Animation/Simulation

Steering

```
/*Steering module execution sequence animation*/
#include <GL/glut.h>
#include <stdlib.h>
#include<stdio.h>
#include<time.h>
#include<math.h>
#define VMR 0.0005// Vertical motion rate
#define HMR 0.0005// Horizontal moition rate
#define SurfaceThreshold 1
#define MaxAngle 90
#define MastRate 0.05
#define UP 1
static GLfloat MastAngle;
static GLfloat FromLatitude; // The begining latitude
static GLfloat FromLongitude; //The beginning longitude
static GLfloat MastMotion = 0.0; // Motion of the mast
static GLfloat Slope; // To get the slope of the line in LINE steeringmode
static GLfloat LatLongDiff; // The constant in the equation of a line for LINE steermode
static GLfloat ToLatitude;
static GLfloat ToLongitude;
float TimeOfOperation, TotalTime;
int SteerMode;
int z=0;
int Steering, count;
float temp1, temp2, square;
float temp3, number, ExitTime,squareroot;
int i = 0 ; //Used for abort signal
int j =0, k =0, n =0, l =1;//j for lat loop, k for long loop, l for LINE loop
int Lat = 0, begin, o = 0, m = 0, DevState__MastState;
time_t start, start1, start2, start3, start4;
time_t end, end1, end2, end3, end4;
double elapsed;
FILE *STOutput, *SOutput, *SteerOutput, *StAngle;

/*Initialization module*/
void init(void)
{

  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_FLAT);
}

/*Display module*/

void display(void)
{
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f(0.0, 0.4, 0.8);
  glRectf(-8.0, -4.0, 8.0, 2.0);
  glPushMatrix();
  glTranslatef (-1.0, 0.0, 0.0);
  glTranslatef (1.0, 0.0, 0.0);
```

```
  glTranslatef(FromLongitude, FromLatitude, 0.0);
  glTranslatef(FromLongitude, FromLatitude, 0.0);
  glPushMatrix();
  glScalef (2.0, 0.4, 1.0);
  glColor3f(0.0, 0.7, 0.4);
  glutSolidCube (1.0);
  glPopMatrix();
  glTranslatef (0.0, 0.1, 0.0);
  glTranslatef(0.08*FromLongitude, MastMotion, 0.0);
  glRotatef(0.0, 0.0, 0.0, 1.0);
  glRotatef(MastAngle, 0.0, 0.0, 1.0);
  glTranslatef(0.5, MastMotion, 0.0);
  glPushMatrix();
  glScalef (0.8, 0.2, 0.5);
  glColor3f(1.0, 1.0, 0.0);
  glutSolidCube(1.0);
  glPopMatrix();
  glPopMatrix();
  glutSwapBuffers();
}

/*Module to Steer the AUV*/
void Steer(void)
{
        if(begin == 0 )
        {
                time(&start);
                time(&start1);
                time(&start2);
                time(&start3);
                time(&start4);
                begin++;
        }

        if (Steering == 1)
         {

         /*AUV moving horizontally when FromLongitude is lesser than ToLongitude*/

                if ((FromLongitude < ToLongitude) && (k == 0))
                {

                 /*AUV for LINE steering mode to get slope*/
                        if ((SteerMode == 1) && (n == 0))
                        {
                                Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
                                LatLongDiff = FromLatitude - Slope*FromLongitude;
                                n++; l--;
                        }

                        /*AUV for LINE steering mode */

                        if (SteerMode == 1 && l ==0)
                        {
                                FromLongitude = FromLongitude + HMR;
                                FromLatitude = Slope*FromLongitude + LatLongDiff;
```

```
                                        MastMotion = 0.008*FromLongitude;
                                        j = 1;

                                        if(difftime(end,start1) == 2)

                                        {
                                                printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start1), FromLatitude, FromLongitude);
                                                time(&start1);
                                        }

                                        if(ToLongitude <= FromLongitude)
                                        {
                                        glutIdleFunc(NULL);
                                                k =1;
                                                l = 1;
                                                printf("Time = %f Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);
                                        }
                                        glutPostRedisplay();
                                }

                        }

                /*AUV moving horizontally when FromLongitude is greater than ToLongitude*/

                        if ((ToLongitude < FromLongitude ) && (k == 0) )
                        {
                                /*AUV for LINE steering mode to get slope*/

                                if (SteerMode == 1 && n == 0)
                                {
                                        Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
                                        LatLongDiff = FromLatitude - Slope*FromLongitude;
                                        n++; l--;
                                }

                                /*AUV for LINE steering mode*/

                                if (SteerMode == 1 && l == 0 )
                                {
                                        FromLongitude = FromLongitude - HMR;
                                        FromLatitude = Slope*FromLongitude + LatLongDiff;
                                        MastMotion = 0.008*FromLongitude;
                                        j = 1;

                                        if((difftime(end,start2)) == 2)
                                        {
                                                printf("Time = %f, Steermode LINE Lat = %f, Long =
%f\n",difftime(end,start2), FromLatitude, FromLongitude);
                                                time(&start2);
                                        }

                                        if(ToLongitude >= FromLongitude )
                                        {
                                        glutIdleFunc(NULL);
```

```
                                        k =1;
                                        l = 1;
                                        printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);

                                }
                                glutPostRedisplay();
                        }

                }


                /*AUV moving Vertically when FromLatitude is lesser than ToLatitude */

                if (FromLatitude < ToLatitude && j == 0 )
                {
                                FromLatitude = FromLatitude + VMR;
                                MastMotion = 0.008*FromLatitude;

                                if(difftime(end,start3) == 2)
                                {
                                        if (SteerMode == 1)
                                                printf("Time = %f Steermode LINE Lat:%f Long =
%f\n", difftime(end,start3), FromLatitude, FromLongitude);
                                        time(&start3);
                                }

                                if((FromLatitude >= ToLatitude) && (FromLongitude ==
ToLongitude))
                                {

                                        glutIdleFunc(NULL);
                                        j = 1;
                                        k = 1;
                                        printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);
                                }

                                glutPostRedisplay();

                        }


                /*AUV moving Vertically when FromLatitude is greater than ToLatitude */


                if (FromLatitude > ToLatitude && j == 0 )
                {

                        FromLatitude = FromLatitude - VMR;
                        MastMotion = 0.008*FromLatitude;

                        if(difftime(end,start4)== 2)
                        {
                                if (SteerMode == 1)
```

```
                                                printf("Time = %f Steermode LINE Lat:%f Long =
%f\n",difftime(end,start4), FromLatitude, FromLongitude);
                                        time(&start4);;

                                }

                                if((FromLatitude <= ToLatitude) && (FromLongitude == ToLongitude))
                                {
                                        glutIdleFunc(NULL);
                                        j = 1;
                                        k = 1;
                                        printf("Time = %f, Steermode LINE %f, %f\n",difftime(end,start),
FromLatitude, FromLongitude);
                                }

                                glutPostRedisplay();
                        }

                }

                time(&end);
 if(k == 1 && j == 1)
                {
                        if(count == 0)
                        {
                                SOutput = fopen( "C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "w"
);
                                fprintf( SOutput, "%f\n", FromLatitude );
                                fprintf( SOutput, "%f\n", FromLongitude );
                                fclose(SOutput);
                                SteerOutput = fopen(
"C:\\Research\\Animation\\Temp\\Mission\\SteerOutput.txt", "w" );
                                fprintf( SteerOutput, "%f\n", FromLatitude );
                                fprintf( SteerOutput, "%f\n", FromLongitude );
                                fprintf(SteerOutput, "%f\n", difftime(end, start));
                                fclose(SteerOutput);

                                STOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "r");
                                fscanf(STOutput,"%f\n", &TotalTime);
                                fclose(STOutput);

                                TotalTime = TotalTime + difftime(end, start);

                                STOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "w");
                                fprintf(STOutput, "%f\n", TotalTime);
                                fclose(STOutput);
                                count = 1;
                                exit(0);
                        }
                }

}

void Abort(void)
{
/*AUV moving up to take GPSFix (Control within GPSFix module)*/
```

```
        /*Take value of SurfaceThreshold from Uppaal file */
        if (FromLatitude < SurfaceThreshold && o ==0)
        {
                FromLatitude = FromLatitude + VMR;
                MastMotion = 0.008*FromLatitude;

                if(FromLatitude == SurfaceThreshold)
                {
                glutIdleFunc(NULL);
                        o = 1;
                }
                glutPostRedisplay();
        }

        if (FromLatitude >= SurfaceThreshold && m == 0)
        {

                MastAngle = MastAngle + MastRate;

                /*Rate to raise mast when AUV is on water surface rate valur from Uppaaal*/
                while (MastAngle>MaxAngle)  /*Take value of MastAngle from Uppaal*/
                {
                        DevState__MastState = UP;
                        m = 1;
                }

                glutPostRedisplay();
        }

}


/*Projection module*/

void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(70.0, (GLfloat) w/(GLfloat) h, 10.0, 4.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef (0.0, 0.0, -5.0);
}

/*Keyboard function*/

void keyboard (unsigned char key, int x, int y)
{
  switch (key) {

    case 'a':
                {
                        i = 1;
                          glutIdleFunc(Abort);
                          printf("MISSION ABORTED\n");
```

```
                    }
                    break;

    case 27:
                    {

      exit(0);
                    }
                    break;
    default:
                    break;
  }
}

/*Mouse action*/

void mouse(int button, int state, int x, int y)

{
        switch(button)
                {
                        case GLUT_LEFT_BUTTON:

                                if (state == GLUT_DOWN && i == 0.0)
                                {
                                        glutIdleFunc(Steer);
                                }
                                break;
                }
}

FILE *SInput, *StInput, *SAngle;
int main(int argc, char** argv)
{
        glutInit(&argc, argv);

        /*Open file for reading input*/

        SInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\SteerInput.txt", "r");
        if( SInput == NULL )
    printf( "The file 'SteerInput.txt' was not opened\n" );// File failed to open
        else
    printf( "The file 'SteerInput.txt' was opened\n" );// File opened

        fseek( SInput, 0L, SEEK_SET );
        fscanf(SInput, "%d\n", &Steering);//Get the order is Steering or not
        fscanf(SInput, "%d\n", &SteerMode);//Mode of Steering
        fscanf(SInput, "%f\n", &ToLatitude);//Get the destined Latitude
        fscanf(SInput, "%f\n", &ToLongitude);//Get the destned Longitude
  fclose(SInput);
        printf("ToLat = %f, ToLong = %f\n",ToLatitude, ToLongitude);

        if (ToLatitude < -1.5)
        {
                ToLatitude = -1.5;
                printf("ToLatitude value change to -1.5 as depth below that is dangerous\n");
```

```
	}
	if (ToLatitude > 1.0)
	{
		ToLatitude = 1.0;
		printf("ToLatitude value cannot be more than 1.0 the surface\n");
	}

	StInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "r");
	fscanf(SInput, "%f\n", &FromLatitude);//Get the current Latitude
	fscanf(SInput, "%f\n", &FromLongitude);//Get the current Longitude
	printf("Lat = %f, Long = %f\n",FromLatitude, FromLongitude);

	if (FromLatitude < -1.5)
	{
		FromLatitude = -1.5;
		printf("FromLatitude value change to -1.5 as depth below that is dangerous\n");
	}

	if (FromLatitude > 1.0)
	{
		FromLatitude = 1.0;
		printf("FromLatitude value cannot be more than 1.0 the surface\n");
	}

	temp1 = FromLatitude - ToLatitude;
	temp2 = FromLongitude - ToLongitude;
	temp3 = temp1*temp1 + temp2*temp2;
	squareroot = sqrt(temp3);
	printf("%f\n", squareroot);

	SAngle = fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "r");
	fscanf(SAngle, "%f\n", &MastAngle);
	if (MastAngle > 0 && FromLatitude < 1.0)
	{
		MastAngle = 0.0;
		printf("Mast is not raised as not on surface\n");
	}
	fclose(SAngle);
	if (MastAngle > 0 && FromLatitude == 1.0)
	{
		MastAngle = 0.0;
		printf("Lower Mast before going below surface of water\n");
	}

glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
glutInitWindowSize (500, 500);
glutInitWindowPosition (100, 100);
glutCreateWindow (argv[0]);
init ();
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
glutMouseFunc(mouse);

glutMainLoop();
```

```
  return 0;
}
```

Loiter

```
/*Loiter module execution sequence animation*/

#include <GL/glut.h>
#include <stdlib.h>
#include<stdio.h>
#include<math.h>
#include<time.h>

#define VMR 0.001//Rate to move up depeding
#define HMR 0.001 //Rate to move horizontally
#define MastRate 0.02
#define DriftRate 0.001 //To assign a value to water current given as drift
#define StartPoint -1.0 //Keeping a record of start point
#define Steering 1//Checks whether steering mode received or not
#define LINE 1 //Steering mode direct to go straight to the point following a line
#define HOVER 1
#define CIRCLE 2
#define NONE 0
#define PI 3.14
#define Steering 1
#define LoiterTO 50
#define Loiter_TimeToLoiterPt_FCN 10
#define SurfaceThreshold 1
#define MaxAngle 90
#define DOWN 0
#define UP 1

static GLfloat Helm__DistanceToPoint;
static GLfloat FromLatitude , ToLatitude; // The begining latitude
static GLfloat FromLongitude, ToLongitude; //The beginning longitude
static GLfloat MastMotion = 0.0; // Motion of the mast
static GLfloat i = 0; //Used for abort signal
static GLfloat Slope; // To get the slope of the line in direct steeringmode
static GLfloat SteerMode = LINE;//Assigning mode to steering
static GLfloat LatLongDiff; // The constant in the equation of a line for direct steermode
static GLfloat Step;
static GLfloat ToWP__LoiterType = 1;
static GLfloat Loitering = 0;
static GLfloat First = 0;
static GLfloat r;
static GLfloat Theta;
static GLfloat base;
static LoiterDone = 0, MastAngle = 0;
static GLfloat m = 0;
static GLfloat Side;
static GLfloat TimeInState;
static GLfloat Helm =0;
static GLfloat HoverPoint;
int j =0, k =0, n =0, l =1, p=0, o = 0;//j for lat loop, k for long loop, l for direct loop, p for indirect
int NumPoints = 1, count = 0, Hove = 0, begin = 0, DevState__MastState = DOWN, numCount = 0;
float TotalTime;
```

```
float Time, Timer;
float TimeIncr;
time_t start, start1, start2, start3, start4;
time_t end, end1, end2, end3;
FILE *LOutput, *LoOutput, *LTOutput;
void init(void)
{

  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_FLAT);
}

void display(void)
{
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f(0.0, 0.4, 0.8);
  glRectf(-8.0, -4.0, 8.0, 2.0);
  glPushMatrix();
  glTranslatef (-1.0, 0.0, 0.0);
  glTranslatef (1.0, 0.0, 0.0);
  glTranslatef(FromLongitude, FromLatitude, 0.0);
  glTranslatef(FromLongitude, FromLatitude, 0.0);
  glPushMatrix();
  glScalef (2.0, 0.4, 1.0);
  glColor3f(0.0, 0.7, 0.4);
  glutSolidCube (1.0);
  glPopMatrix();
  glTranslatef (0.0, 0.1, 0.0);
  glTranslatef(0.08*FromLongitude, MastMotion, 0.0);
  glRotatef(0.0, 0.0, 0.0, 1.0);
  glTranslatef(0.5, MastMotion, 0.0);
  glPushMatrix();
  glScalef (0.8, 0.2, 0.5);
  glColor3f(1.0, 1.0, 0.0);
  glutSolidCube(1.0);
  glPopMatrix();
  glPopMatrix();
  glutSwapBuffers();
}


void Loiter(void)
{
        if(begin == 0)
        {
                time(&start);
                time(&start1);
                begin++;
        }

        if ((NumPoints<2 && LoiterTO == 0 || ToWP__LoiterType == NONE) && Loitering == 0)
        {
                Loitering = 1;
                glutIdleFunc(NULL);
                printf("No time to Loiter \n");
```

```
	}

	if ((NumPoints<2 && LoiterTO>0) || ToWP__LoiterType == HOVER && LoiterDone == 0)
	{
		if (count == 0)
		{
			HoverPoint = FromLongitude;
			printf("HOVERING, HoverPoint = %f, Hove \n", HoverPoint);
			count++;
		}

		if (FromLongitude < (HoverPoint + 1.0) && Hove == 0 && LoiterDone == 0)
		{
			FromLongitude = FromLongitude + 0.001;

			if (FromLongitude >= HoverPoint + 1.0)
			{
				++Hove;
				printf("Hove in fwd= %d, Time Hovering = %f\n", Hove,
difftime(end1, start));

			}
			if ((difftime(end1, start) >= (LoiterTO-Loiter_TimeToLoiterPt_FCN-20)))
			{
				LoiterDone = 1;

			}
		}

		if (FromLongitude > HoverPoint  && Hove == 1 && LoiterDone == 0)
		{
			FromLongitude = FromLongitude - 0.001;

			if (FromLongitude <= HoverPoint)
			{
				Hove = 0;
				printf("Hove in bwd= %d Time Hovering = %f \n",Hove,
difftime(end1, start));
			}
			if ((difftime(end1, start) >= (LoiterTO-Loiter_TimeToLoiterPt_FCN-20)))
			{
				LoiterDone = 1;

			}
		}

			glutPostRedisplay();
	}
	time(&end1);

	if (NumPoints>=2 && LoiterTO>0 && ToWP__LoiterType == CIRCLE && (difftime(end2,
start)<=LoiterTO-Loiter_TimeToLoiterPt_FCN) && Helm ==0)
	{
		Step = 360/NumPoints;
		for (m=0; m<Step; m=m+1)
		{
```

```
if (First == 0)
{

	First = 1;
	r = pow(pow((FromLatitude-ToLatitude),2) + pow((FromLongitude-
ToLongitude),2), 0.5);
	Helm__DistanceToPoint = r;
	if (Helm__DistanceToPoint <= 0.25)
	{
		glutIdleFunc(NULL);
		printf("Distance to point is less than 20 m\n");
		Helm++;
	}
	Side = pow(pow((FromLatitude-ToLatitude-r),2) +
pow((FromLongitude-ToLongitude),2), 0.5);
	Theta = 2*asin(Side/(2*r));
	printf("r=%f, Theta = %f\n",r, Theta);
	FromLongitude = -r * cos(Theta) + ToLongitude;
	FromLatitude = -r * sin(Theta) + ToLatitude;

	glutPostRedisplay();
	m++;
	printf("FromLat = %f, FromLong = %f, m = %f\n", FromLatitude,
FromLongitude, m);
}

if (m>0 && LoiterDone == 0)
{

	Theta = Theta + 0.00001;
	FromLongitude = -r * cos(Theta)+ ToLongitude;
	FromLatitude = -r * sin(Theta)+ ToLatitude;

	if (difftime(end2, start1) == 2)
	{

		printf("TimeInState = %f, FromLat = %f, FromLong = %f\n",
difftime(end2, start), FromLatitude, FromLongitude);
		time(&start1);
	}

	if (difftime(end2, start)>=LoiterTO-Loiter_TimeToLoiterPt_FCN)

	{
		LoiterDone = 1;
	}

	if(FromLatitude > 1.0)
	{
		FromLatitude = 1;

	}
	if(FromLatitude < -1.5)
	{
```

```
                                        FromLatitude = -1.5;

                                }

                                glutPostRedisplay();
                        }

                }

        }
        time(&end2);


        if (Steering == 1 && LoiterDone == 1 && Helm == 0)
        {
                if(begin == 0 )
                {
                        time(&start);
                        time(&start1);
                        time(&start2);
                        time(&start3);
                        time(&start4);
                        begin++;
                }

        /*AUV moving horizontally when FromLongitude is lesser than ToLongitude*/

                if ((FromLongitude < ToLongitude) && (k == 0))
                {

                 /*AUV for LINE steering mode to get slope*/
                        if ((SteerMode == LINE) && (n == 0))
                        {
                                Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
                                LatLongDiff = FromLatitude - Slope*FromLongitude;
                                n++; l--;
                        }

                        /*AUV for LINE steering mode */

                        if (SteerMode == LINE && l ==0)
                        {
                                FromLongitude = FromLongitude + HMR;
                                FromLatitude = Slope*FromLongitude + LatLongDiff;
                                MastMotion = 0.008*FromLongitude;
                                j = 1;

                                if(difftime(end,start1) == 2)

                                {
                                        printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start1), FromLatitude, FromLongitude);
                                        time(&start1);
                                }

                                if(ToLongitude <= FromLongitude)
```

```
                                        {
                                        glutIdleFunc(NULL);
                                                k =1;
                                                l = 1;
                                                printf("Time = %f Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);
                                        }
                                        glutPostRedisplay();
                                }

                        }


                /*AUV moving horizontally when FromLongitude is greater than ToLongitude*/

                        if ((ToLongitude < FromLongitude ) && (k == 0) )
                        {
                                /*AUV for LINE steering mode to get slope*/

                                if (SteerMode == LINE && n == 0)
                                {
                                        Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
                                        LatLongDiff = FromLatitude - Slope*FromLongitude;
                                        n++; l--;
                                }

                                /*AUV for LINE steering mode*/

                                if (SteerMode == LINE && l == 0 )
                                {
                                        FromLongitude = FromLongitude - HMR;
                                        FromLatitude = Slope*FromLongitude + LatLongDiff;
                                        MastMotion = 0.008*FromLongitude;
                                        j = 1;

                                        if((difftime(end,start2)) == 2)
                                        {
                                                printf("Time = %f, Steermode LINE Lat = %f, Long =
%f\n",difftime(end,start2), FromLatitude, FromLongitude);
                                                time(&start2);
                                        }

                                        if(ToLongitude >= FromLongitude )
                                        {
                                        glutIdleFunc(NULL);
                                                k =1;
                                                l = 1;
                                                printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);

                                        }
                                        glutPostRedisplay();

                                }

                        }
```

```
		/*AUV moving Vertically when FromLatitude is lesser than ToLatitude */

		if (FromLatitude < ToLatitude && j == 0 )
		{
				FromLatitude = FromLatitude + VMR;
				MastMotion = 0.008*FromLatitude;

				if(difftime(end,start3) == 2)
				{
					if (SteerMode == LINE)
						printf("Time = %f Steermode LINE Lat:%f Long =
%f\n", difftime(end,start3), FromLatitude, FromLongitude);
					time(&start3);
				}

				if((FromLatitude >= ToLatitude) && (FromLongitude ==
ToLongitude))
				{

					glutIdleFunc(NULL);
					j = 1;
					k = 1;
					printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);
				}

				glutPostRedisplay();

			}


		/*AUV moving Vertically when FromLatitude is greater than ToLatitude */

		if (FromLatitude > ToLatitude && j == 0 )
		{

			FromLatitude = FromLatitude - VMR;
			MastMotion = 0.008*FromLatitude;

			if(difftime(end,start4)== 2)
			{
				if (SteerMode == LINE)
					printf("Time = %f Steermode LINE Lat:%f Long =
%f\n",difftime(end,start4), FromLatitude, FromLongitude);
				time(&start4);;

			}

			if((FromLatitude <= ToLatitude) && (FromLongitude == ToLongitude))
			{
				glutIdleFunc(NULL);
				j = 1;
				k = 1;
				printf("Time = %f, Steermode LINE %f, %f\n",difftime(end,start),
FromLatitude, FromLongitude);
			}
```

```
                        glutPostRedisplay();

                }

}

        time(&end);
        if(k == 1 && j == 1)
        {
                if(numCount == 0)
                {
                        LOutput = fopen( "C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "w"
);
                        fprintf( LOutput, "%f\n", FromLatitude );
                        fprintf( LOutput, "%f\n", FromLongitude );
                        fclose(LOutput);

                        LoOutput = fopen( "C:\\Research\\Animation\\Temp\\Mission\\SteerOutput.txt",
"w" );
                        fprintf( LoOutput, "%f\n", FromLatitude );
                        fprintf( LoOutput, "%f\n", FromLongitude );
                        fprintf(LoOutput, "%f\n", difftime(end, start));
                        fclose(LoOutput);

                        LTOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "r");
                        fscanf(LTOutput,"%f\n", &TotalTime);
                        fclose(LTOutput);

                        TotalTime = TotalTime + difftime(end, start);

                        LTOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "w");
                        fprintf(LTOutput, "%f\n", TotalTime);
                        fclose(LTOutput);
                        count = 1;
                        exit(0);
                }
        }


}

void Abort(void)
{
/*AUV moving up to take GPSFix (Control within GPSFix module)*/
        /*Take value of SurfaceThreshold from Uppaal file */
        if (FromLatitude < SurfaceThreshold && o ==0)
        {
                FromLatitude = FromLatitude + VMR;
                MastMotion = 0.008*FromLatitude;

                if(FromLatitude == SurfaceThreshold)
                {
                glutIdleFunc(NULL);
                        o = 1;
                }
                glutPostRedisplay();
```

```
		}

		if (FromLatitude >= SurfaceThreshold && m == 0)
		{

			MastAngle = MastAngle + MastRate;
			/*Rate to raise mast when AUV is on water surface rate valur from Uppaaal*/
			while (MastAngle>MaxAngle)  /*Take value of MastAngle from Uppaal*/
			{
				DevState__MastState = UP;
				m = 1;
			}
			glutPostRedisplay();
		}
}

void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(70.0, (GLfloat) w/(GLfloat) h, 10.0, 4.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef (0.0, 0.0, -5.0);
}

void keyboard (unsigned char key, int x, int y)
{
  switch (key) {

    case 'a':
                 {
                        i = 1;
                  glutIdleFunc(Abort);
                  printf("Mission Aborted\n");
                 }
                break;

    case 27:
      exit(0);
      break;
    default:
      break;
  }
}
void mouse(int button, int state, int x, int y)

{
		switch(button)
			{
				case GLUT_LEFT_BUTTON:
					if (state == GLUT_DOWN && i == 0.0)
					{
						glutIdleFunc(Loiter);
```

```
                                        }
                                        break;
                        }
}

FILE *LInput, *LoInput;

int main(int argc, char** argv)
{
  glutInit(&argc, argv);

  LInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "r");
            if( LInput == NULL )
    printf( "The file 'Position.txt' was not opened\n" );// File failed to open
            else
    printf( "The file 'Position.txt' was opened\n" );// File opened

            fseek( LInput, 0L, SEEK_SET );
            fscanf(LInput, "%f\n", &FromLatitude);//Get the destined Latitude
            fscanf(LInput, "%f\n", &FromLongitude);//Get the destned Longitude
   fclose(LInput);

            LoInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\LoiterInput.txt", "r");
            if( LoInput == NULL )
    printf( "The file 'LoiterInput.txt' was not opened\n" );// File failed to open
            else
    printf( "The file 'LoiterInput.txt' was opened\n" );// File opened

            fseek( LoInput, 0L, SEEK_SET );
            fscanf(LoInput, "%f\n", &ToLatitude);//Get the destined Latitude
            fscanf(LoInput, "%f\n", &ToLongitude);//Get the destned Longitude
   fclose(LoInput);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMouseFunc(mouse);
  glutMainLoop();
  return 0;
}
```

DeviceCommander

Device module to raise AUV to water surface

```
/*Device commander module used to raise AUV to surface  sequence execution animation*/
#include <GL/glut.h>
#include <stdlib.h>
#include<stdio.h>
#include<time.h>
```

```
static GLfloat FromLatitude; // The begining latitude
static GLfloat FromLongitude; //The beginning longitude
static GLfloat MastMotion = 0.0; // Motion of the mast
static GLfloat Slope; // To get the slope of the line in direct steeringmode
static GLfloat LatLongDiff; // The constant in the equation of a line for direct steermode
static GLfloat ToLatitude = 1.0;
static GLfloat ToLongitude;
static GLfloat VMR;
static GLfloat HMR;
int SteerMode;
int z=0, begin = 0;
int Steering, count;
int i = 0; //Used for abort signal
int j =0, k =0, n =0, l =1, p=0;//j for lat loop, k for long loop, l for direct loop, p for indirect
int Lat = 0;
float Timed, TimeOfOperation;
float Time, Timer;
float temp1, temp2;
float temp3, number, TotalTime;
FILE *DCGOutput, *DCGOut, *DCGTOutput;
time_t start, start1, start2, start3, start4;
time_t end;

/*Initialization module*/

void init(void)
{

  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_FLAT);
}

/*Display module*/

void display(void)
{
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f(0.0, 0.4, 0.8);
  glRectf(-8.0, -4.0, 8.0, 2.0);
  glPushMatrix();
  glTranslatef (-1.0, 0.0, 0.0);
  glTranslatef (1.0, 0.0, 0.0);
  glTranslatef(FromLongitude, FromLatitude, 0.0);
  glTranslatef(FromLongitude, FromLatitude, 0.0);
  glPushMatrix();
  glScalef (2.0, 0.4, 1.0);
  glColor3f(0.0, 0.7, 0.4);
  glutSolidCube (1.0);
  glPopMatrix();
  glTranslatef (0.0, 0.1, 0.0);
  glTranslatef(0.08*FromLongitude, MastMotion, 0.0);
  glRotatef(0.0, 0.0, 0.0, 1.0);
  glTranslatef(0.5, MastMotion, 0.0);
  glPushMatrix();
  glScalef (0.8, 0.2, 0.5);
  glColor3f(1.0, 1.0, 0.0);
```

```
  glutSolidCube(1.0);
  glPopMatrix();
  glPopMatrix();
  glutSwapBuffers();
}

/*Module to Steer the AUV*/
void Steer(void)
{

        if (Steering == 1)
         {
                if(FromLatitude == 1)
                {
                        glutIdleFunc(NULL);
                        printf("Already at surface\n");
                        j=1;
                        k=1;
                        exit(0);
                }
                if(begin == 0)
                {
                        time(&start);
                        time(&start1);
                        time(&start2);
                        time(&start3);
                        begin++;
                }

         /*AUV moving horizontally when FromLongitude is lesser than ToLongitude*/

                if ((FromLongitude < ToLongitude) && (k == 0))
                {

                 /*AUV for LINE steering mode to get slope*/

                        if ((SteerMode == 1) && (n == 0))
                        {
                                Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
                                LatLongDiff = FromLatitude - Slope*FromLongitude;
                                n++; l--;
                        }

                        /*AUV for LINE steering mode */

                        if (SteerMode == 1 && l ==0)
                        {
                                FromLongitude = FromLongitude + HMR;
                                FromLatitude = Slope*FromLongitude + LatLongDiff;
                                MastMotion = 0.008*FromLongitude;
                                j = 1;
                                if (difftime(end, start1) == 2)

                                {
                                        printf("Time = %f, Steermode LINE %f, %f\n",difftime(end,
start1), FromLatitude, FromLongitude);
```

```
                                        time(&start1);
                                }

                                if(ToLongitude <= FromLongitude)
                                {
                                glutIdleFunc(NULL);
                                        k =1;
                                        l = 1;
                                        printf("Time = %f Steermode LINE %f, %f\n",difftime(end,
start),FromLatitude, FromLongitude);
                                }
                                glutPostRedisplay();
                        }

                }

        /*AUV moving horizontally when FromLongitude is greater than ToLongitude*/
                if ((ToLongitude < FromLongitude ) && (k == 0) )
                {

                        /*AUV for LINE steering mode to get slope*/

                        if (SteerMode == 1 && n == 0)
                        {
                                Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
                                LatLongDiff = FromLatitude - Slope*FromLongitude;
                                n++; l--;
                        }

                        /*AUV for LINE steering mode*/

                        if (SteerMode == 1 && l == 0 )
                        {
                                FromLongitude = FromLongitude - HMR;
                                FromLatitude = Slope*FromLongitude + LatLongDiff;
                                MastMotion = 0.008*FromLongitude;
                                j = 1;
                                if (difftime(end, start2) == 2)

                                {
                                        printf("Time = %f, Steermode LINE Lat = %f, Long =
%f\n",difftime(end, start), FromLatitude, FromLongitude);
                                        time(&start2);
                                }

                                if(ToLongitude >= FromLongitude )
                                {
                                glutIdleFunc(NULL);
                                        k =1;
                                        l = 1;
                                        printf("Time = %f, Steermode LINE %f, %f\n",difftime(end,
start), FromLatitude, FromLongitude);

                                }
                                glutPostRedisplay();
                        }
```

```
		}

		/*AUV moving Vertically when FromLatitude is lesser than ToLatitude */

		if (FromLatitude < ToLatitude && j == 0 )
		{
				FromLatitude = FromLatitude + VMR;
				MastMotion = 0.008*FromLatitude;

				if(difftime(end, start3) == 2)
				{
					printf("Time = %f Steermode LINE Lat:%f Long = %f\n",
difftime(end, start), FromLatitude, FromLongitude);
					time(&start3);
				}

				if((FromLatitude >= ToLatitude) && (FromLongitude ==
ToLongitude))
				{
					glutIdleFunc(NULL);
					j = 1;
					printf("Time = %f, Steermode LINE %f, %f\n",difftime(end,
start), FromLatitude, FromLongitude);
				}
				else if (FromLatitude>= ToLatitude)
				{
					j = 1;
					printf("Stop Lat Time = %f, Steermode LINE %f,
%f\n",difftime(end, start), FromLatitude, FromLongitude);
				}
				glutPostRedisplay();
			}


		/*AUV moving Vertically when FromLatitude is greater than ToLatitude */

		if (FromLatitude > ToLatitude && j == 0 )
		{
			FromLatitude = FromLatitude - VMR;
			MastMotion = 0.008*FromLatitude;

			if (difftime(end, start4) == 2)
			{
				printf("Time = %f Steermode LINE Lat:%f Long = %f\n",difftime(end,
start), FromLatitude, FromLongitude);
				time(&start4);

			}

			if((FromLatitude <= ToLatitude) && (FromLongitude == ToLongitude))
			{
				glutIdleFunc(NULL);
				j = 1;
				printf("Time = %f, Steermode LINE %f, %f\n",difftime(end, start),
FromLatitude, FromLongitude);
			}
```

```
                                else if (FromLatitude <= ToLatitude)
                                {
                                        j = 1;
                                        printf("Stop Lat motion Time = %f, Steermode LINE %f,
%f\n",difftime(end, start), FromLatitude, FromLongitude);
                                }
                                glutPostRedisplay();

                        }

                }

                if(k == 1 && j == 1)
                {
                        if(count == 0)
                        {
                                DCGOutput = fopen( "C:\\Research\\Animation\\Temp\\Mission\\Position.txt",
"w" );
                                fprintf( DCGOutput, "%f\n", FromLatitude );
                                fprintf( DCGOutput, "%f\n", FromLongitude );
                                fclose(DCGOutput);

                                DCGOut = fopen( "C:\\Research\\Animation\\Temp\\Mission\\DCGOutput.txt",
"w" );
                                fprintf( DCGOut, "%f\n", FromLatitude );
                                fprintf( DCGOut, "%f\n", FromLongitude );
                                fprintf(DCGOut, "%f\n", difftime(end, start));
                                fclose(DCGOut);

                                DCGTOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt",
"r");
                                fscanf(DCGTOutput,"%f\n", &TotalTime);
                                fclose(DCGTOutput);

                                TotalTime = TotalTime + difftime(end, start);

                                DCGTOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt",
"w");
                                fprintf(DCGTOutput, "%f\n", TotalTime);
                                fclose(DCGTOutput);
                                count = 1;
                                exit(0);
                        }

                }
                time(&end);
}

/*Projection module*/

void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(70.0, (GLfloat) w/(GLfloat) h, 10.0, 4.0);
```

```
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef (0.0, 0.0, -5.0);
}

/*Keyboard function*/

void keyboard (unsigned char key, int x, int y)
{
  switch (key) {

    case 'a':
                        {
                                i = 1;
                          glutIdleFunc(NULL);
                          printf("MISSION ABORTED\n");
                        }
                       break;

    case 27:
                        {

      exit(0);
                        }
                       break;
    default:
                       break;
  }
}

/*Mouse action*/

void mouse(int button, int state, int x, int y)

{
        switch(button)
                {
                        case GLUT_LEFT_BUTTON:
                                if (state == GLUT_DOWN && i == 0.0)
                                {
                                        glutIdleFunc(Steer);

                                }
                                break;

                }
}
FILE *DCGInput, *DCGOutput, *DCGIn;
int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        /*Open file for reading input*/
        DCGInput = fopen("C:\\Research\\Animation\\Temp\\DC_Gen_GoToSurface\\DCGInput.txt",
"r");
        if( DCGInput == NULL )
    printf( "The file 'DCGInput.txt' was not opened\n" );// File failed to open
```

```
        else
    printf( "The file 'DCGInput.txt' was opened\n" );// File opened
        fseek( DCGInput, 0L, SEEK_SET );
        fscanf(DCGInput, "%f\n", &Timed);//Check whether Timed operation or not (0 (untimed) or 1
(Timed) )
        fscanf(DCGInput, "%f\n", &TimeOfOperation);//Time of operation
        fscanf(DCGInput, "%d\n", &Steering);//Get the order is Steering or not
        fscanf(DCGInput, "%d\n", &SteerMode);//Get the mode of Steering
        //fscanf(SInput, "%f\n", &ToLatitude);//Get the destined Latitude
        fscanf(DCGInput, "%f\n", &ToLongitude);//Get the destned Longitude
   fclose(DCGInput);
        if (Timed == 0)
        {
                VMR = 0.001;
                HMR = 0.001;
        }
        DCGIn = fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "r");
        fseek( DCGIn, 0L, SEEK_SET );
        fscanf(DCGIn, "%f\n", &FromLatitude);
        fscanf(DCGIn, "%f\n", &FromLongitude);
        fclose(DCGIn);
        printf("Lat = %f, Long = %f\n",FromLatitude, FromLongitude);

  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMouseFunc(mouse);
  DCGOutput = fopen( "C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "w" );
  fprintf( DCGOutput, "%f\n", ToLatitude );
  fprintf( DCGOutput, "%f\n", ToLongitude );
  fclose( DCGOutput );
  glutMainLoop();
  return 0;
}
```

Device module to lower AUV from water surface

```
/*Device commander module to bring AUV below surface of water sequence execution animation*/

#include <GL/glut.h>
#include <stdlib.h>
#include<stdio.h>
#include<time.h>
#define DriftRate 0.001 //To assign a value to water current given as drift
static GLfloat FromLatitude; // The begining latitude
static GLfloat FromLongitude; //The beginning longitude
static GLfloat MastMotion = 0.0; // Motion of the mast
int i = 0; //Used for abort signal
int j =0, k =0, n =0, l =1, p=0;//j for lat loop, k for long loop, l for direct loop, p for indirect
int Lat = 0;
static GLfloat Slope; // To get the slope of the line in direct steeringmode
```

```
static GLfloat LatLongDiff; // The constant in the equation of a line for direct steermode
static GLfloat ToLatitude;
static GLfloat ToLongitude;
static GLfloat VMR;
static GLfloat HMR;
float Timed, TimeOfOperation;
int SteerMode;
int z=0;
int Steering, count, begin = 0;
float Time, Timer;
float temp1, temp2;
float temp3, number, TotalTime;
time_t start, start1, start2, start3, start4;
time_t end, end1, end2, end3;
FILE *DCSOutput, *DCSOut, *DCSTOutput;
/*Initialization module*/

void init(void)
{

  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_FLAT);
}

/*Display module*/

void display(void)
{
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f(0.0, 0.4, 0.8);
  glRectf(-8.0, -4.0, 8.0, 2.0);
  glPushMatrix();
  glTranslatef (-1.0, 0.0, 0.0);
  glTranslatef (1.0, 0.0, 0.0);
  glTranslatef(FromLongitude, FromLatitude, 0.0);
  glTranslatef(FromLongitude, FromLatitude, 0.0);
  glPushMatrix();
  glScalef (2.0, 0.4, 1.0);
  glColor3f(0.0, 0.7, 0.4);
  glutSolidCube (1.0);
  glPopMatrix();
  glTranslatef (0.0, 0.1, 0.0);
  glTranslatef(0.08*FromLongitude, MastMotion, 0.0);
  glRotatef(0.0, 0.0, 0.0, 1.0);
  glTranslatef(0.5, MastMotion, 0.0);
  glPushMatrix();
  glScalef (0.8, 0.2, 0.5);
  glColor3f(1.0, 1.0, 0.0);
  glutSolidCube(1.0);
  glPopMatrix();
  glPopMatrix();
  glutSwapBuffers();
}

/*Module to Steer the AUV*/
void Steer(void)
```

```
{
        if (FromLatitude < 1 && j == 0 && p == 0 )
        {
                printf("Already under water\n");
                glutIdleFunc(NULL);
                p++;
                exit(0);

        }

        if(begin == 0 )
        {
                time(&start);
                time(&start1);
                time(&start2);
                time(&start3);
                time(&start4);
                begin++;
        }

        if (Steering == 1 && p == 0 )
         {

         /*AUV moving horizontally when FromLongitude is lesser than ToLongitude*/

                if ((FromLongitude < ToLongitude) && (k == 0))
                {

                 /*AUV for LINE steering mode to get slope*/
                        if ((SteerMode == 1) && (n == 0))
                        {
                                Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
                                LatLongDiff = FromLatitude - Slope*FromLongitude;
                                n++; l--;
                        }

                        /*AUV for LINE steering mode */

                        if (SteerMode == 1 && l ==0)
                        {
                                FromLongitude = FromLongitude + HMR;
                                FromLatitude = Slope*FromLongitude + LatLongDiff;
                                MastMotion = 0.008*FromLongitude;
                                j = 1;

                                if(difftime(end,start1) == 2)

                                {
                                        printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start1), FromLatitude, FromLongitude);
                                        time(&start1);
                                }

                                if(ToLongitude <= FromLongitude)
                                {
```

```
                                        glutIdleFunc(NULL);
                                                k =1;
                                                l = 1;
                                                printf("Time = %f Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);
                                        }
                                        glutPostRedisplay();
                                }

                        }


                /*AUV moving horizontally when FromLongitude is greater than ToLongitude*/
                        if ((ToLongitude < FromLongitude ) && (k == 0) )
                        {
                                /*AUV for LINE steering mode to get slope*/
                                if (SteerMode == 1 && n == 0)
                                {
                                        Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
                                        LatLongDiff = FromLatitude - Slope*FromLongitude;
                                        n++; l--;
                                }

                                /*AUV for LINE steering mode*/

                                if (SteerMode == 1 && l == 0 )
                                {
                                        FromLongitude = FromLongitude - HMR;
                                        FromLatitude = Slope*FromLongitude + LatLongDiff;
                                        MastMotion = 0.008*FromLongitude;
                                        j = 1;

                                        if((difftime(end,start2)) == 2)
                                        {
                                                printf("Time = %f, Steermode LINE Lat = %f, Long =
%f\n",difftime(end,start2), FromLatitude, FromLongitude);
                                                time(&start2);
                                        }

                                        if(ToLongitude >= FromLongitude )
                                        {
                                        glutIdleFunc(NULL);
                                                k =1;
                                                l = 1;
                                                printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);

                                        }
                                        glutPostRedisplay();

                                }

                        }


                        /*AUV moving Vertically when FromLatitude is lesser than ToLatitude */
```

```
		if (FromLatitude < ToLatitude && j == 0 )
		{
				FromLatitude = FromLatitude + VMR;
				MastMotion = 0.008*FromLatitude;
				if(difftime(end,start3) == 2)
				{
					if (SteerMode == 1)
						printf("Time = %f Steermode LINE Lat:%f Long =
%f\n", difftime(end,start3), FromLatitude, FromLongitude);
					time(&start3);
				}

				if((FromLatitude >= ToLatitude) && (FromLongitude ==
ToLongitude))
				{
					glutIdleFunc(NULL);
					j = 1;
					k = 1;
					printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);
				}

				glutPostRedisplay();

			}


		/*AUV moving Vertically when FromLatitude is greater than ToLatitude */

		if (FromLatitude > ToLatitude && j == 0 )
		{

			FromLatitude = FromLatitude - VMR;
			MastMotion = 0.008*FromLatitude;
			if(difftime(end,start4)== 2)
			{
				if (SteerMode == 1)
					printf("Time = %f Steermode LINE Lat:%f Long =
%f\n",difftime(end,start4), FromLatitude, FromLongitude);
				time(&start4);;
			}

			if((FromLatitude <= ToLatitude) && (FromLongitude == ToLongitude))
			{
				glutIdleFunc(NULL);
				j = 1;
				k = 1;
				printf("Time = %f, Steermode LINE %f, %f\n",difftime(end,start),
FromLatitude, FromLongitude);
			}

			glutPostRedisplay();
		}


	}
```

```
		time(&end);
		if(k == 1 && j == 1)
		{
			if(count == 0)
			{
				DCSOutput = fopen( "C:\\Research\\Animation\\Temp\\Mission\\Position.txt",
"w" );
				fprintf( DCSOutput, "%f\n", FromLatitude );
				fprintf( DCSOutput, "%f\n", FromLongitude );
				fclose(DCSOutput);

				DCSOut = fopen( "C:\\Research\\Animation\\Temp\\Mission\\DCSOutput.txt",
"w" );
				fprintf( DCSOut, "%f\n", FromLatitude );
				fprintf( DCSOut, "%f\n", FromLongitude );
				fprintf(DCSOut, "%f\n", difftime(end, start));
				fclose(DCSOut);

				DCSTOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt",
"r");
				fscanf(DCSTOutput,"%f\n", &TotalTime);
				fclose(DCSTOutput);

				TotalTime = TotalTime + difftime(end, start);

				DCSTOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt",
"w");
				fprintf(DCSTOutput, "%f\n", TotalTime);
				fclose(DCSTOutput);
				count = 1;
				exit(0);
			}
		}
}

/*Projection module*/

void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(70.0, (GLfloat) w/(GLfloat) h, 10.0, 4.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef (0.0, 0.0, -5.0);
}

/*Keyboard function*/

void keyboard (unsigned char key, int x, int y)
{
  switch (key) {

    case 'a':
                  {
```

```
                                i = 1;
                                  glutIdleFunc(NULL);
                                  printf("MISSION ABORTED\n");
                         }
                        break;

    case 27:
                         {

      exit(0);
                         }
                        break;
    default:
                        break;
  }
}

/*Mouse action*/

void mouse(int button, int state, int x, int y)

{
        switch(button)
                {
                        case GLUT_LEFT_BUTTON:
                                if (state == GLUT_DOWN && i == 0.0)
                                {
                                        glutIdleFunc(Steer);

                                }
                                break;

                }
}
FILE *DCCPInput, *DCCInput;
int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        /*Open file for reading input*/
        DCCInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "r");
        if( DCCInput == NULL )
    printf( "The file 'Position.txt' was not opened\n" );// File failed to open
        else
    printf( "The file 'Position.txt' was opened\n" );// File opened
        fseek( DCCInput, 0L, SEEK_SET );
        fscanf(DCCInput, "%f\n", &FromLatitude);//Get the destined Latitude
        fscanf(DCCInput, "%f\n", &FromLongitude);//Get the destned Longitude
        printf("FromLat = % f, FromLong = %f\n",FromLatitude, FromLongitude);
  fclose(DCCInput);
        DCCPInput =
fopen("C:\\Research\\Animation\\Temp\\DC_Gen_ComeOffSurface\\DCCPInput.txt", "r");
        if( DCCPInput == NULL )
    printf( "The file 'DCCPInput.txt' was not opened\n" );// File failed to open
        else
    printf( "The file 'DCCPInput.txt' was opened\n" );// File opened
```

```
        fseek( DCCPInput, 0L, SEEK_SET );
        fscanf(DCCPInput, "%f\n", &Timed);//Check whether Timed operation or not (0 (untimed) or 1
(Timed) )
        fscanf(DCCPInput, "%f\n", &TimeOfOperation);//Time of operation
        fscanf(DCCPInput, "%d\n", &Steering);//Get the order is Steering or not
        fscanf(DCCPInput, "%d\n", &SteerMode);//Get the mode of Steering
        fscanf(DCCPInput, "%f\n", &ToLatitude);//Get the destined Latitude
        fscanf(DCCPInput, "%f\n", &ToLongitude);//Get the destned Longitude
   fclose(DCCPInput);
        if (Timed == 0)
        {
                VMR = 0.001;
                HMR = 0.001;
        }

  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMouseFunc(mouse);
  glutMainLoop();
  return 0;
}
```

Device module to raise mast

```
/*Device commander module to raise mast sequence execution animation*/

#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

#define MastRate 0.02 /*Rate to raise mast*/
#define UP 0.0 /*To check is mast raised*/
#define DONE 1.0 /*To check is GPSFix taken*/
#define DOWN 1.0 /*To check is mast lowered*/
#define Yes 1.0 /*If need to return to starting point after GPSFix is taken*/
#define No 0.0 /*If not to return to starting point*/
#define MaxAngle 90.0 /*The angle mast needs to be raised to take GPSFix*/
#define WaitToTakeGPSFix 25.0 /*Time to wait till GPSFix is taken*/
#define FAILED 1.0 /*Indicating GPSFix failed*/
static GLfloat Slope; // To get the slope of the line in direct steeringmode
static GLfloat LatLongDiff; // The constant in the equation of a line for direct steermode
static GLfloat FromLatitude ;
static GLfloat StartLongitude;
static GLfloat StartLatitude;
static GLfloat DevState__MastState=DOWN;
static GLfloat DevState__GPSFixState=0;
static GLfloat GPSOrd__ReturnToStart=1;
static GLfloat Helm__DistanceToPoint=-1;
```

```
static GLfloat GoToSurfaceTO=2500;
static GLfloat RaiseMastTO=50;
static GLfloat TakeFixTO=0;
static GLfloat NumFailed=0;
static GLfloat MastAngle = 0.0, MastDown =0.0, Angle=0.0;
static GLfloat MastMotion = 0.0;
static int shoulder = 0.0, elbow = 0.0;
static GLfloat Wait = 0.0;
static GLfloat TotalTimeLowerMast = 0.0, TotalTimeDown =0.0,
TotalTimeUp = 0.0, TotalTimeMastUp = 0.0;
static GLfloat SurfaceThreshold = 1;
static GLfloat Drift;
static GLfloat FromLongitude;
static GLfloat TimeInState = 0.0;
static GLfloat GPSFix = 0.0;
static GLfloat ReachSurface = 0.0;
static GLfloat TimeToChangeDirection = 0.0;
int DisplayOnce = 0;
int j =0, k =0, n =0, l =1, p=0;//j for lat loop, k for long loop, l for direct loop, p for indirect
int count = 0, numCount = 0, begin = 0;
int i =0.0, m = 0, o = 0, MastLowered = 0.0;
int MastRaised = 0, MastIsRaised = 0 ;
float Time, Timer, MissionTime, RaiseMastTime, LowerMast;
float TotalTime;
time_t start, start1;
time_t end;
FILE *DCROutput, *DCRPOutput, *DCRPTOutput;

void init(void)
{
 glClearColor (0.0, 0.0, 0.0, 0.0);
 glShadeModel (GL_FLAT);
}
void display(void)
{
 glClear (GL_COLOR_BUFFER_BIT);
 glColor3f(0.0, 0.4, 0.8);
 glRectf(-8.0, -4.0, 8.0, 2.0);
 glPushMatrix();
  glTranslatef (-1.0, 0.0, 0.0);
  glRotatef ((GLfloat) shoulder, 0.0, 0.0, 1.0);
  glTranslatef (1.0, 0.0, 0.0);
 glTranslatef(FromLongitude, FromLatitude, 0.0);
 glTranslatef(FromLongitude, FromLatitude, 0.0);
 glPushMatrix();
 glScalef (2.0, 0.4, 1.0);
 glColor3f(0.0, 0.7, 0.4);
 glutSolidCube (1.0);
 glPopMatrix();
 glTranslatef (0.0, 0.1, 0.0);
 glTranslatef(0.08*FromLongitude, MastMotion, 0.0);
 glRotatef(0.0, 0.0, 0.0, 1.0);
 glRotatef(MastAngle, 0.0, 0.0, 1.0);
 glTranslatef(0.5, MastMotion, 0.0);
 glPushMatrix();
 glScalef (0.8, 0.2, 0.5);
```

```
 glColor3f(1.0, 1.0, 0.0);
 glutSolidCube(1.0);
 glPopMatrix();
 glPopMatrix();
 glutSwapBuffers();
}
void MoveUp(void)
{
	if(begin == 0)
	{
		time(&start);
		begin++;
	}
	if(FromLatitude < 1 && numCount == 0)
	{
		printf("AUV is not at surface mast cannot be raised\n");
		numCount++;
	}

/*Once AUV is at the surface of water module to raise mast*/
	if (FromLatitude >= SurfaceThreshold)
	{

		if(begin == 1)
		{
			time(&start1);
			begin++;
		}

		MastAngle = MastAngle + MastRate;

		if(difftime(end, start1) == 2)
		{
			RaiseMastTime = RaiseMastTime + difftime(end, start1);
			printf("MissionTime = %f, RaiseMastTime = %f, MastAngle = %f\n",
difftime(end, start), RaiseMastTime, MastAngle);
			time(&start1);
		}

		/*Rate to raise mast when AUV is on water surface rate valur from Uppaaal*/
		if (MastAngle>MaxAngle)  /*Take value of MastAngle from Uppaal*/
		{
			DevState__MastState = UP;

			glutIdleFunc(NULL);

		}
		glutPostRedisplay();
	}
	time(&end);
	if(DisplayOnce == 0 && DevState__MastState == UP)
			{
				printf("MissionTime = %f, RaiseMastTime = %f, MastAngle = %f\n",
difftime(end, start), RaiseMastTime, MastAngle);
```

```
                                        DCROutput = fopen(
"C:\\Research\\Animation\\Temp\\Mission\\Angle.txt", "w" );
                                        fprintf( DCROutput, "%f\n", MastAngle );
                                        fclose( DCROutput );

                                        DCRPOutput =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "w" );
                                        fprintf( DCRPOutput, "%f\n", FromLatitude );
                                        fprintf( DCRPOutput, "%f\n", FromLongitude );
                                        fclose( DCRPOutput );

                                        DCRPTOutput =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "r" );
                                        fscanf(DCRPTOutput, "%f\n", &TotalTime);
                                        fclose(DCRPTOutput);
                                        TotalTime = TotalTime + difftime(end, start);

                                        DCRPTOutput =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "w" );
                                        fprintf(DCRPTOutput, "%f\n", TotalTime);
                                        fclose(DCRPTOutput);

                                        DisplayOnce++;
                                        exit(0);
                                }
}

void reshape (int w, int h)
{
 glViewport (0, 0, (GLsizei) w, (GLsizei) h);
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
gluPerspective(70.0, (GLfloat) w/(GLfloat) h, 10.0, 4.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
 glRotatef(MastAngle, 0.0, 0.0, 1.0);
 glTranslatef (0.0, 0.0, -5.0);
}
void keyboard (unsigned char key, int x, int y)
{
  switch (key)
{
     case 's':
        shoulder = (shoulder + 5) % 360;
        glutPostRedisplay();
        break;
    case 'S':
        shoulder = (shoulder - 5) % 360;
        glutPostRedisplay();
        break;
case 'a':
                        {
                                i = 1;
                                  glutIdleFunc(NULL);
                        }
                       break;
```

```
    case 27:
      exit(0);
      break;
    default:
      break;
  }
}
void mouse(int button, int state, int x, int y)
{
switch(button)
{
case GLUT_LEFT_BUTTON:
if (state == GLUT_DOWN && i == 0.0)
{
glutIdleFunc(MoveUp);
}
break;
}
}

FILE *DCRInput;
int main(int argc, char** argv)
{
glutInit(&argc, argv);
DCRInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "r");
if( DCRInput == NULL )
    printf( "The file 'Position.txt' was not opened\n" );// File failed to open
          else
    printf( "The file 'Position.txt' was opened\n" );// File opened

          fseek( DCRInput, 0L, SEEK_SET );
          fscanf(DCRInput, "%f\n", &FromLatitude);//Get the destined Latitude
          fscanf(DCRInput, "%f\n", &FromLongitude);//Get the destned Longitude
          printf("FromLat = %f FromLong = %f\n", FromLatitude, FromLongitude);

glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
 glutInitWindowSize (500, 500);
glutInitWindowPosition (100, 100);
glutCreateWindow (argv[0]);
init ();
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
glutMouseFunc(mouse);
glutMainLoop();
return 0;
}
```

Device module to lower mast

```
/*Device commander module to lower mast sequence execution animation*/

#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
```

```
#define VMR 0.001 /*Rate to move up depeding upon time*/
#define MastRate 0.02 /*Rate to raise mast*/
#define UP 0.0 /*To check is mast raised*/
#define DONE 1.0 /*To check is GPSFix taken*/
#define DOWN 1.0 /*To check is mast lowered*/
#define DriftRate 0.001 /*To assign a value to water current given as drift*/
#define Yes 1.0 /*If need to return to starting point after GPSFix is taken*/
#define No 0.0 /*If not to return to starting point*/
#define MaxAngle 90.0 /*The angle mast needs to be raised to take GPSFix*/
#define Depth 0.0 /*The depth at which the AUV dives to*/
#define WaitToTakeGPSFix 25.0 /*Time to wait till GPSFix is taken*/
#define FAILED 1.0 /*Indicating GPSFix failed*/
#define NotInTime 1.0
#define HMR 0.001 //Rate to move horizontally
#define Steering 1//Checks whether steering mode received or not
#define Direct 1 //Steering mode direct to go straight to the point following a line
#define NotDirect 2 //Steering mode indirect
#define SurfaceThreshold 1
static GLfloat ToLongitude; // The begining latitude
static GLfloat Slope; // To get the slope of the line in direct steeringmode
static GLfloat SteerMode = Direct;//Assigning mode to steering
static GLfloat LatLongDiff; // The constant in the equation of a line for direct steermode
static GLfloat FromLatitude = 1;
static GLfloat FromLongitude = 0;
static GLfloat StartLongitude;
static GLfloat StartLatitude;
static GLfloat DevState__MastState=DOWN;
static GLfloat DevState__GPSFixState=0;
static GLfloat GPSOrd__ReturnToStart=1;
static GLfloat Helm__DistanceToPoint=-1;
static GLfloat GoToSurfaceTO=2500;
static GLfloat RaiseMastTO=50;
static GLfloat TakeFixTO=0;
static GLfloat NumFailed=0;
static GLfloat MastAngle = 90.0, MastDown =0.0, Angle=0.0;
static GLfloat MastMotion = 0.0;
static int shoulder = 0.0, elbow = 0.0;
static GLfloat Wait = 0.0;
static GLfloat TotalTimeLowerMast = 0.0, TotalTimeDown =0.0,
TotalTimeUp = 0.0, TotalTimeMastUp = 0.0;
static GLfloat ToLatitude = SurfaceThreshold;
static GLfloat Drift;
static GLfloat TimeInState = 0.0;
static GLfloat GPSFix = 0.0;
static GLfloat ReachSurface = 0.0;
static GLfloat TimeToChangeDirection = 0.0;
int MastRaised = 0, MastIsRaised = 0, numCount = 0;
int begin = 0;
int j =0, k =0, n =0, l =1, p=0;//j for lat loop, k for long loop, l for direct loop, p for indirect
int i =0.0, m = 0, o = 0, MastLowered = 0.0;
float Time, Timer, MissionTime, RaiseMastTime ;
float TotalTime;
double LowerMast;
time_t start, start1;
time_t end;
FILE *DCLPOutput, *DCLMOutput;
```

```
void init(void)
{
 glClearColor (0.0, 0.0, 0.0, 0.0);
 glShadeModel (GL_FLAT);
}
void display(void)
{
 glClear (GL_COLOR_BUFFER_BIT);
 glColor3f(0.0, 0.4, 0.8);
 glRectf(-8.0, -4.0, 8.0, 2.0);
 glPushMatrix();
 glTranslatef (-1.0, 0.0, 0.0);
 glRotatef ((GLfloat) shoulder, 0.0, 0.0, 1.0);
 glTranslatef (1.0, 0.0, 0.0);
 glTranslatef(FromLongitude, FromLatitude, 0.0);
 glTranslatef(FromLongitude, FromLatitude, 0.0);
 glPushMatrix();
 glScalef (2.0, 0.4, 1.0);
 glColor3f(0.0, 0.7, 0.4);
 glutSolidCube (1.0);
 glPopMatrix();
 glTranslatef (0.0, 0.1, 0.0);
 glTranslatef(0.08*FromLongitude, MastMotion, 0.0);
 glRotatef(0.0, 0.0, 0.0, 1.0);
 glRotatef(MastAngle, 0.0, 0.0, 1.0);
 glTranslatef(0.5, MastMotion, 0.0);
 glPushMatrix();
 glScalef (0.8, 0.2, 0.5);
 glColor3f(1.0, 1.0, 0.0);
 glutSolidCube(1.0);
 glPopMatrix();
 glPopMatrix();
 glutSwapBuffers();
}
void MoveUp(void)
{
        if(begin == 0)
        {
                time(&start);
                begin++;
        }

        if(MastAngle < 0 && numCount == 0)
        {
                printf("Mast not raised\n");
                numCount++;
        }
 /***Lower mast when GPSFix is taken (Control passed to Launch module)***/
                if ((FromLatitude >= 1 && MastAngle > 0))
                {
                        if(begin == 1)
                        {
                                time(&start1);
                                begin++;
                        }
```

```
                        MastAngle = MastAngle - 1.5*MastRate;/*Lowering mast rate specified in
uppaal*/

                        if (difftime(end, start1) == 1)
                        {
                                LowerMast = LowerMast + difftime(end, start1);
                                printf ("MissionTime = %f, LowerMast = %f, MastAngle =
%f\n",difftime(end, start), LowerMast, MastAngle);
                                time(&start1);
                        }
                                if (MastAngle < 0.0)
                        {
                                DevState__MastState = DOWN;
                                glutIdleFunc(NULL);
                                m =1;
                        }
                 glutPostRedisplay();

                }
                time(&end);

                if (m == 1)
                {

                                DCLPOutput =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Angle.txt", "w" );
                                fprintf( DCLPOutput, "%f\n", MastAngle );
                                fclose( DCLPOutput );
                                DCLMOutput =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "r" );
                                fseek( DCLMOutput, 0L, SEEK_SET );
                                fscanf(DCLMOutput, "%f\n", &TotalTime);
                                fclose(DCLMOutput);
                                TotalTime = TotalTime + difftime(end, start);

                                DCLMOutput =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "w" );
                                fseek( DCLMOutput, 0L, SEEK_SET );
                                fprintf(DCLMOutput, "%f\n", TotalTime);
                                fclose(DCLMOutput);
                                exit(0);
                }
}
void reshape (int w, int h)
{
 glViewport (0, 0, (GLsizei) w, (GLsizei) h);
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
gluPerspective(70.0, (GLfloat) w/(GLfloat) h, 10.0, 4.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef (0.0, 0.0, -5.0);
}
void keyboard (unsigned char key, int x, int y)
{
  switch (key)
```

```
{
     case 's':
        shoulder = (shoulder + 5) % 360;
        glutPostRedisplay();
        break;
    case 'S':
        shoulder = (shoulder - 5) % 360;
        glutPostRedisplay();
        break;
case 'a':
                        {
                                i = 1;
                                  glutIdleFunc(NULL);
                        }
                       break;
     case 27:
        exit(0);
        break;
     default:
        break;
  }
}
void mouse(int button, int state, int x, int y)
{
switch(button)
{
case GLUT_LEFT_BUTTON:
if (state == GLUT_DOWN && i == 0.0)
{
glutIdleFunc(MoveUp);
}
break;
}
}

FILE *DCLPInput, *DCLAInput;
int main(int argc, char** argv)
{
            glutInit(&argc, argv);
            DCLPInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "r");
            if( DCLPInput == NULL )
     printf( "The file 'Position.txt' was not opened\n" );// File failed to open
            else
     printf( "The file 'Position.txt' was opened\n" );// File opened

            fseek( DCLPInput, 0L, SEEK_SET );
            fscanf(DCLPInput, "%f\n", &FromLatitude);//Get the destined Latitude
            fscanf(DCLPInput, "%f\n", &FromLongitude);//Get the destned Longitude
            printf("FromLat = %f FromLong = %f\n", FromLatitude, FromLongitude);
            fclose(DCLPInput);
            DCLAInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Angle.txt", "r");
            fseek( DCLAInput, 0L, SEEK_SET );
            fscanf(DCLAInput, "%f\n", &MastAngle);
            fclose(DCLAInput);

            glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
```

```
        glutInitWindowSize (500, 500);
        glutInitWindowPosition (100, 100);
        glutCreateWindow (argv[0]);
        init ();
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutKeyboardFunc(keyboard);
        glutMouseFunc(mouse);
        glutMainLoop();
        return 0;
}
```

GPSFixer

```
/*GPSFixer module sequence execution animation*/

#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define VMR 0.001 /*Rate to move up depeding upon time*/
#define MastRate 0.05 /*Rate to raise mast*/
#define UP 0.0 /*To check is mast raised*/
#define NOTDONE 0.0
#define DONE 1.0 /*To check is GPSFix taken*/
#define DOWN 1.0 /*To check is mast lowered*/
#define DriftRate 0.001 /*To assign a value to water current given as drift*/
#define Yes 1.0 /*If need to return to starting point after GPSFix is taken*/
#define No 0.0 /*If not to return to starting point*/
#define MaxAngle 90.0 /*The angle mast needs to be raised to take GPSFix*/
#define Depth 0.0 /*The depth at which the AUV dives to*/
#define WaitToTakeGPSFix 25.0 /*Time to wait till GPSFix is taken*/
#define FAILED 1.0 /*Indicating GPSFix failed*/
#define NotInTime 1.0
#define HMR 0.001 //Rate to move horizontally
#define Steering 1//Checks whether steering mode received or not
#define LINE 1 //Steering mode direct to go straight to the point following a line
#define SurfaceThreshold 1
static GLfloat Slope; // To get the slope of the line in direct steeringmode
static GLfloat SteerMode = LINE;//Assigning mode to steering
static GLfloat LatLongDiff; // The constant in the equation of a line for direct steermode
static GLfloat FromLatitude;
static GLfloat FromLongitude;
static GLfloat ToLatitude = SurfaceThreshold;
static GLfloat ToLongitude;
static GLfloat StartLongitude;
static GLfloat StartLatitude;
static GLfloat DevState__MastState=DOWN;
static GLfloat GPSOrd__ReturnToStart = 0;
static GLfloat Helm__DistanceToPoint=-1;
static GLfloat GoToSurfaceTO=500;
static GLfloat RaiseMastTO=50;
static GLfloat TakeFixTO=0;
static GLfloat NumFailed=0;
static GLfloat MastAngle = 0.0, MastDown =0.0, Angle=0.0;
static GLfloat MastMotion = 0.0;
```

```
static int shoulder = 0.0, elbow = 0.0;
static GLfloat TotalTimeLowerMast = 0.0, TotalTimeDown =0.0,
TotalTimeUp = 0.0, TotalTimeMastUp = 0.0;
static GLfloat TimeInState = 0.0;
static GLfloat GPSFix = 0.0;
static GLfloat ReachSurface = 0.0;
static GLfloat TimeToChangeDirection = 0.0;
int MastRaised = 0, MastIsRaised = 0, AUVDown = 0 ;
int j =0, k =0, n =0, l =1, CountNumber = 0; p=0, iteration = 0;//j for lat loop, k for long loop, l for direct
loop, p for indirect
int i =0.0, m = 0, o = 0, MastLowered = 0.0, count = 0, begin = 0, numCount;
int DevState__GPSFixState = NOTDONE;
float Time, Timer, MissionTime;
float TimeOfMission, TotalTime, Timed;
double TimeToRise = 0,RaiseMastTime =0, Wait = 0, LowerMast = 0;
time_t start, start1, start2, start3, start4, start5, start6, start7, start8, start9, start10;
time_t end, end1, end2, end3, end4, end5, end6;
FILE *GOutput, *GPSOutput, *GTOutput;

/*Initialization module*/
void init(void)
{
 glClearColor (0.0, 0.0, 0.0, 0.0);
 glShadeModel (GL_FLAT);
}

/*Display module*/
void display(void)
{
        glClear (GL_COLOR_BUFFER_BIT);
        glColor3f(0.0, 0.4, 0.8);
        glRectf(-8.0, -4.0, 8.0, 2.0);
        glPushMatrix();
        glTranslatef (-1.0, 0.0, 0.0);
        glRotatef ((GLfloat) shoulder, 0.0, 0.0, 1.0);
        glTranslatef (1.0, 0.0, 0.0);
        glTranslatef(FromLongitude, FromLatitude, 0.0);

/*AUV moving down when mast is lowered after GPSFix is taken (Control within Launch  module)*/
        if((DevState__MastState == DOWN && FromLatitude > Depth && AUVDown == 0 &&
MastRaised == 1) || ReachSurface == NotInTime && AUVDown == 0)
        {
                if(begin == 3)
                {
                        time(&start5);
                        begin++;
                }

                FromLatitude = FromLatitude - VMR;
                MastMotion = 0.008*MastMotion;
                if (difftime(end5, start5) == 2)
                {
                        printf("Time = %f, Lat = %f, Long = %f\n", difftime(end5, start5),
FromLatitude, FromLongitude);
                        time(&start5);
                }
```

```
		glutPostRedisplay();
	}
	time(&end5);

	if (FromLatitude <= Depth && AUVDown == 0)
	{
		AUVDown = 1;
	}

	if (DevState__MastState == DOWN && MastRaised == 1 && GPSOrd__ReturnToStart == No
&& FromLatitude <= Depth)
	{
		glutIdleFunc(NULL);
		if(count == 0)
			{
				GOutput = fopen(
"C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "w" );
				fprintf( GOutput, "%f\n", FromLatitude );
				fprintf( GOutput, "%f\n", FromLongitude );
				fclose(GOutput);

				GPSOutput = fopen(
"C:\\Research\\Animation\\Temp\\Mission\\GPSFixOutput.txt", "w" );
				fprintf( GPSOutput, "%f\n", FromLatitude );
				fprintf( GPSOutput, "%f\n", FromLongitude );
				fprintf(GPSOutput, "%f\n", difftime(end5, start));
				fprintf(GPSOutput, "Time to Rise to water %f\n", TimeToRise);
				fprintf(GPSOutput, "Time to raise mast%f\n", RaiseMastTime);
				fprintf(GPSOutput, "Time to lower mast%f\n",LowerMast);
				fclose(GPSOutput);

				GTOutput =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "r");
				fscanf(GTOutput,"%f\n", &TotalTime);
				fclose(GTOutput);

				TotalTime = TotalTime + difftime(end5, start);

				GTOutput =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "w");
				fprintf(GTOutput, "%f\n", TotalTime);
				fclose(GTOutput);
				count = 1;
				exit(0);
			}
	}

if(DevState__MastState == DOWN && AUVDown == 1 && MastRaised == 1 &&
GPSOrd__ReturnToStart == Yes)
{
	ToLongitude = StartLongitude;
	ToLatitude = StartLatitude;
	/*Check whether to Return back to the starting point (Control wihtin GPSFix)
						If need to return to starting point control is passed to Steering
module*/
```

```
if(begin == 4 )
{
        time(&start6);
        begin++;
}
if (Steering == 1)
 {

 /*AUV moving horizontally when FromLongitude is lesser than ToLongitude*/

        if ((FromLongitude < ToLongitude) && (k == 0))
        {

         /*AUV for Direct steering mode to get slope*/

                if ((SteerMode == 1) && (n == 0))
                {
                Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
                LatLongDiff = FromLatitude - Slope*FromLongitude;
                n++; l--;
                }

                /*AUV for Direct steering mode */

                if (SteerMode == 1 && l ==0)
                {
                        FromLongitude = FromLongitude + HMR ;
                        FromLatitude = Slope*FromLongitude + LatLongDiff;
                        MastMotion = 0.008*FromLongitude;
                        j = 1;

                        if (difftime(end, start6) == 2)
                        {
                                printf("Time = %f, Steermode Direct Lat= %f  Long: %f\n",
difftime(end6, start6),FromLatitude, FromLongitude);
                                time(&start6);
                        }

                        if(FromLongitude >= ToLongitude)
                        {
                        glutIdleFunc(NULL);
                                k =1;
                                l = 1;
                                o = 1;
                                printf("Time = %f, Steermode Direct Lat= %f  Long: %f\n",
difftime(end6, start6),FromLatitude, FromLongitude);

                        }
                        glutPostRedisplay();
                }


        }

 /*AUV moving horizontally when FromLongitude is greater than ToLongitude*/
```

```
if ((FromLongitude > ToLongitude) && (k == 0))
{

        /*AUV for Direct steering mode to get slope*/

        if ((SteerMode == 1) && (n == 0))
        {
        Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
        LatLongDiff = FromLatitude - Slope*FromLongitude;
        n++; l--;
        }

        /*AUV for Direct steering mode*/

        if (SteerMode == 1 && l == 0)
        {
                FromLongitude = FromLongitude - HMR ;
                FromLatitude = Slope*FromLongitude + LatLongDiff;
                MastMotion = 0.008*FromLongitude;
                j = 1;

                if (difftime(end, start6) == 2)
                {
                        printf("Time = %f, Steermode Direct Lat= %f  Long:
%f\n",difftime(end, start6),FromLatitude, FromLongitude);
                        time(&start6);
                }
                if(FromLongitude <= ToLongitude)
                {
                glutIdleFunc(NULL);
                        k =1;
                        l = 1;
                        o = 1;
                        printf("MissionTime = %f, Steermode Direct Lat= %f  Long:
%f\n", difftime(end, start),FromLatitude, FromLongitude);

                }
                glutPostRedisplay();
        }

}

/*AUV moving Vertically when FromLatitude is lesser than ToLatitude */

if (FromLatitude < ToLatitude && j == 0)
{

                FromLatitude = FromLatitude + VMR;
                MastMotion = 0.008*FromLatitude;

                if (difftime(end, start6) == 2)
                {
                        if (SteerMode == 1)
                        {
```

```
                                                printf("Time = %f Steermode Direct Lat:%f Long =
%f\n", difftime(end, start6), FromLatitude, FromLongitude);
                                                time(&start6);
                                        }


                                }
                                if((FromLatitude >= ToLatitude) )
                                {
                                        glutIdleFunc(NULL);
                                        j = 1;
                                        o = 1;
                                        k = 1;
                                        printf("MissionTime = %f, Steermode Direct Lat = %f, Long
= %f\n",difftime(end, start), FromLatitude, FromLongitude);
                                }
                                glutPostRedisplay();



        }


        /*AUV moving Vertically when FromLatitude is greater than ToLatitude */

        if (FromLatitude > ToLatitude && j == 0)
        {

                FromLatitude = FromLatitude - VMR;
                MastMotion = 0.008*FromLatitude;

                if (difftime(end, start6) == 2)
                        {
                                if (SteerMode == 1)
                                {
                                        printf("MissionTime = %f Steermode Direct Lat:%f
Long = %f\n", difftime(end, start6), FromLatitude, FromLongitude);
                                        time(&start6);
                                }


                        }
                if(FromLatitude <= ToLatitude && ToLongitude == FromLongitude)
                {
                        glutIdleFunc(NULL);
                        j = 1;
                        o = 1;
                        k = 1;
                        printf("MissionTime = %f, Steermode NotDirect Lat:%f Long = %f\n",
difftime(end, start), FromLatitude, FromLongitude);
                }

                glutPostRedisplay();
        }
                time(&end);
```

```
                    if(k == 1 && j == 1)
                    {
                              if(count == 0)
                              {

                                        GOutput = fopen(
"C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "w" );
                                        fprintf( GOutput, "%f\n", ToLatitude );
                                        fprintf( GOutput, "%f\n", ToLongitude );
                                        fclose(GOutput);

                                        GPSOutput = fopen(
"C:\\Research\\Animation\\Temp\\Mission\\GPSFixOutput.txt", "w" );
                                        fprintf( GPSOutput, "%f\n", ToLatitude );
                                        fprintf( GPSOutput, "%f\n", ToLongitude );
                                        fprintf(GPSOutput, "%f\n", difftime(end, start));
                                        fprintf(GPSOutput, "Time to Rise to water %f\n", TimeToRise);
                                        fprintf(GPSOutput, "Time to raise mast%f\n", RaiseMastTime);
                                        fprintf(GPSOutput, "Time to lower mast%f\n",LowerMast);
                                        fprintf(GPSOutput, "%f\n",difftime(end, start));
                                        fclose(GPSOutput);

                                        GTOutput =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "r");
                                        fscanf(GTOutput,"%f\n", &TotalTime);
                                        fclose(GTOutput);

                                        TotalTime = TotalTime + difftime(end, start);

                                        GTOutput =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "w");
                                        fprintf(GTOutput, "%f\n", TotalTime);
                                        fclose(GTOutput);
                                        count = 1;
                                        exit(0);
                              }
                    }
          }
}

 glTranslatef(FromLongitude, FromLatitude, 0.0);
 glPushMatrix();
 glScalef (2.0, 0.4, 1.0);
 glColor3f(0.0, 0.7, 0.4);
 glutSolidCube (1.0);
 glPopMatrix();
 glTranslatef (0.0, 0.1, 0.0);
 glTranslatef(0.08*FromLongitude, MastMotion, 0.0);
 glRotatef(0.0, 0.0, 0.0, 1.0);
 glRotatef(MastAngle, 0.0, 0.0, 1.0);
 glTranslatef(0.5, MastMotion, 0.0);
 glPushMatrix();
 glScalef (0.8, 0.2, 0.5);
 glColor3f(1.0, 1.0, 0.0);
 glutSolidCube(1.0);
 glPopMatrix();
```

```
 glPopMatrix();
 glutSwapBuffers();
}

void MoveUp(void)
{
/*AUV moving up to take GPSFix (Control within GPSFix module)*/
        /*Take value of SurfaceThreshold from Uppaal file */
        if(Timed == 1)
        {
                if(TotalTime > TimeOfMission)
                {

                        printf("Time has elapsed\n");
                        glutIdleFunc(NULL);
                        exit(0);
                }
        }

        if(begin == 0)
        {
                time(&start);
                time(&start1);

                begin++;
        }

        if (FromLatitude < SurfaceThreshold && o ==0)
        {
                FromLatitude = FromLatitude + VMR;
                MastMotion = 0.008*FromLatitude;

                if(FromLatitude == SurfaceThreshold)
                {

                        o = 1;
                        glutIdleFunc(NULL);
                }

                glutPostRedisplay();


                if(difftime(end1, start1) == 2)
                {
                        TimeToRise = TimeToRise + difftime(end1, start1);
                        printf("Time = %f, RiseTime = %f, FromLat = %f, FromLong = %f
\n",difftime(end1, start1), TimeToRise, FromLatitude, FromLongitude);
                        time(&start1);
                }

                if (GoToSurfaceTO < TimeToRise)
                {
                        printf("GPSFix failed as couldnt reach surface in time\n");
                        glutIdleFunc(NULL);
                }
        }
```

```
	time(&end1);
	/*Once AUV is at the surface of water module to raise mast*/
	if (FromLatitude >= SurfaceThreshold && m == 0 && DevState__MastState == DOWN)
	{
		if(begin == 1)
		{
			time(&start2);
			begin++;
		}

		MastAngle = MastAngle + MastRate;
		if(difftime(end2, start2) == 1 && Wait == 0)
		{
			RaiseMastTime = RaiseMastTime + difftime(end2, start2);
			printf("Time = %f, RaiseMastTime = %f, MastAngle = %f\n", difftime(end2,
start2), RaiseMastTime, MastAngle);
			time(&start2);
		}
		/*Rate to raise mast when AUV is on water surface rate valur from Uppaaal*/
		if (MastAngle>MaxAngle && iteration == 0 && Wait == 0)  /*Take value of MastAngle
from Uppaal*/
		{
			DevState__MastState = UP;
			printf("Time = %f, RaiseMastTime = %f, MastAngle = %f\n", difftime(end2,
start2), RaiseMastTime, MastAngle);
			iteration++;
		}
		/*To check whether time to actuall raie mast is more than given*/
		if (RaiseMastTime>= RaiseMastTO)
		{
			GPSFix = FAILED;
			glutIdleFunc(NULL);
		}
		glutPostRedisplay();
	}
	time(&end2);

	if (GPSFix == FAILED)
	{
		while(MastAngle > 0.0)
		MastAngle = MastAngle -0.00005*MastRate;/*Lowering mast rate specified in uppaal*/
		if (MastAngle < 0.0 )
		{
			DevState__MastState = DOWN;
			glutIdleFunc(NULL);
		}
		printf("GPSFix FAILED\n");
		glutPostRedisplay();

	}
/****Wait till GPSFix is taken***/

	if(DevState__MastState == UP && iteration == 1)
	{
		time(&start3);
		printf("Wait for 5 seconds to takeGPSFix\n");
```

```
                do
                {
                        time(&end3);

                } while(  difftime(end3, start3) < 5 );

                DevState__GPSFixState = DONE;

                iteration++;
        }
/***Lower mast when GPSFix is taken (Control passed to Launch module)***/
        if (DevState__GPSFixState == DONE)
        {
        if (MastAngle > 0.0 && iteration == 2)
                {
                                if(begin == 2)
                                {
                                        time(&start4);
                                        begin++;
                                }

                                MastAngle = MastAngle - MastRate;

                                if(difftime(end4, start4) == 1)
                                {
                                        LowerMast = LowerMast + difftime(end4, start4);
                                        printf ("Time = %f, LowerMast = %f, MastAngle =
%f\n",difftime(end4, start4), LowerMast, MastAngle);
                                         time(&start4);
                                }
                                if (MastAngle < 0.0)
                                {
                                        DevState__MastState = DOWN;
                                        MastRaised = 1;
                                        AUVDown = 0;
                                        if(Wait > WaitToTakeGPSFix)
                                                printf("GPSFix failed as time exceeded\n");
                                        else
                                                printf("GPSFix is successful, AUVDown = %d,
DevState__MastState = %d MastRaised = %d\n", AUVDown,DevState__MastState, MastRaised);

                                        glutIdleFunc(NULL);
                                        m =1;
                                }
                                glutPostRedisplay();
                }
        time(&end4);
        }

}

void reshape (int w, int h)
{
 glViewport (0, 0, (GLsizei) w, (GLsizei) h);
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
```

```
gluPerspective(70.0, (GLfloat) w/(GLfloat) h, 10.0, 4.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
 glRotatef(MastAngle, 0.0, 0.0, 1.0);
 glTranslatef (0.0, 0.0, -5.0);
}
void keyboard (unsigned char key, int x, int y)
{
   switch (key)
{
      case 's':
         shoulder = (shoulder + 5) % 360;
         glutPostRedisplay();
         break;
     case 'S':
         shoulder = (shoulder - 5) % 360;
         glutPostRedisplay();
         break;
case 'a':
                          {
                                    i = 1;
                                      glutIdleFunc(NULL);
                          }
                         break;
      case 27:
         exit(0);
         break;
      default:
         break;
   }
}
void mouse(int button, int state, int x, int y)
{
switch(button)
{
case GLUT_LEFT_BUTTON:
if (state == GLUT_DOWN && i == 0.0)
{
glutIdleFunc(MoveUp);
}
break;
}
}

FILE *GInput, *GPSFInput, *GPSFixInput;
int main(int argc, char** argv)
{
glutInit(&argc, argv);
GInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "r");
if( GInput == NULL )
      printf( "The file 'Position.txt' was not opened\n" );// File failed to open
            else
      printf( "The file 'Position.txt' was opened\n" );// File opened
            fseek( GInput, 0L, SEEK_SET );
            fscanf(GInput, "%f\n", &FromLatitude);//Get the destined Latitude
            fscanf(GInput, "%f\n", &FromLongitude);//Get the destned Longitude
```

```
        StartLatitude = FromLatitude;
        StartLongitude = FromLongitude;
        GPSFixInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\GPSInput.txt", "r");
        fscanf(GPSFixInput, "%f\n", &Timed);
        fscanf(GPSFixInput, "%f\n", &TimeOfMission);
        fclose(GPSFixInput);
        GPSFInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "r");
        if (GPSFInput == NULL)
                printf("Time not opened\n");
        else
                printf("Time opened\n");
        fscanf(GPSFInput, "%f\n", &TotalTime);
        fclose(GPSFInput);
        printf("Timed = %f Total time = %f Time of mission = %f\n",Timed, TotalTime,
TimeOfMission);
        glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowSize (500, 500);
        glutInitWindowPosition (100, 100);
        glutCreateWindow (argv[0]);
        init ();
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutKeyboardFunc(keyboard);
        glutMouseFunc(mouse);
        glutMainLoop();
        return 0;
}
```

Pause

```
/*Pause module execution sequence animation*/

#include <GL/glut.h>
#include <stdlib.h>
#include<stdio.h>
#include<time.h>
static GLfloat FromLatitude; // The begining latitude
static GLfloat FromLongitude; //The beginning longitude
static GLfloat MastMotion = 0.0; // Motion of the mast
static GLfloat Slope; // To get the slope of the line in direct steeringmode
static GLfloat LatLongDiff; // The constant in the equation of a line for direct steermode
static GLfloat ToLatitude = 1.0;
static GLfloat ToLongitude;
static GLfloat VMR;
static GLfloat HMR;
int i = 0; //Used for abort signal
int j =0, k =0, n =0, l =1, p=0;//j for lat loop, k for long loop, l for direct loop, p for indirect
int Lat = 0;
int SteerMode;
int z=0, begin = 0;
int Steering, count;
float Timed, TimeOfOperation;
float Time, Timer;
float temp1, temp2;
float temp3, number, TotalTime;
time_t start, start1, start2, start3, start4;
```

```
time_t end, time1, time2;
FILE *DCGOutput, *DCGOut, *DCGTOutput;
/*Initialization module*/
void init(void)
{

  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_FLAT);
}

/*Display module*/

void display(void)
{
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f(0.0, 0.4, 0.8);
  glRectf(-8.0, -4.0, 8.0, 2.0);
  glPushMatrix();
  glTranslatef (-1.0, 0.0, 0.0);
  glTranslatef (1.0, 0.0, 0.0);
  glTranslatef(FromLongitude, FromLatitude, 0.0);
  glTranslatef(FromLongitude, FromLatitude, 0.0);
  glPushMatrix();
  glScalef (2.0, 0.4, 1.0);
  glColor3f(0.0, 0.7, 0.4);
  glutSolidCube (1.0);
  glPopMatrix();
  glTranslatef (0.0, 0.1, 0.0);
  glTranslatef(0.08*FromLongitude, MastMotion, 0.0);
  glRotatef(0.0, 0.0, 0.0, 1.0);
  glTranslatef(0.5, MastMotion, 0.0);
  glPushMatrix();
  glScalef (0.8, 0.2, 0.5);
  glColor3f(1.0, 1.0, 0.0);
  glutSolidCube(1.0);
  glPopMatrix();
  glPopMatrix();
  glutSwapBuffers();
}

/*Module to Steer the AUV*/
void Steer(void)
{

        if (Steering == 1)
         {

                if(FromLatitude == 1)
                {
                        glutIdleFunc(NULL);
                        printf("Already at surface\n");
                        j=1;
                        k=1;
                        exit(0);
                }
                if(begin == 0)
```

```
		{
			time(&start);
			time(&start1);
			time(&start2);
			time(&start3);
			begin++;
		}

	/*AUV moving horizontally when FromLongitude is lesser than ToLongitude*/

		if ((FromLongitude < ToLongitude) && (k == 0))
		{

		 /*AUV for LINE steering mode to get slope*/

			if ((SteerMode == 1) && (n == 0))
			{
				Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
				LatLongDiff = FromLatitude - Slope*FromLongitude;
				n++; l--;
			}

			/*AUV for LINE steering mode */

			if (SteerMode == 1 && l ==0)
			{
				FromLongitude = FromLongitude + HMR;
				FromLatitude = Slope*FromLongitude + LatLongDiff;
				MastMotion = 0.008*FromLongitude;
				j = 1;
				if (difftime(end, start1) == 2)

				{
					printf("Time = %f, Steermode LINE %f, %f\n",difftime(end,
start1), FromLatitude, FromLongitude);
					time(&start1);
				}

				if(ToLongitude <= FromLongitude)
				{
				glutIdleFunc(NULL);
					k =1;
					l = 1;
					printf("Time = %f Steermode LINE %f, %f\n",difftime(end,
start),FromLatitude, FromLongitude);
				}
				glutPostRedisplay();
			}

		}

	/*AUV moving horizontally when FromLongitude is greater than ToLongitude*/

		if ((ToLongitude < FromLongitude ) && (k == 0) )
		{
```

```
                        /*AUV for LINE steering mode to get slope*/
                        if (SteerMode == 1 && n == 0)
                        {
                                Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
                                LatLongDiff = FromLatitude - Slope*FromLongitude;
                                n++; l--;
                        }

                        /*AUV for LINE steering mode*/

                        if (SteerMode == 1 && l == 0 )
                        {
                                FromLongitude = FromLongitude - HMR;
                                FromLatitude = Slope*FromLongitude + LatLongDiff;
                                MastMotion = 0.008*FromLongitude;
                                j = 1;
                                if (difftime(end, start2) == 2)

                                {
                                        printf("Time = %f, Steermode LINE Lat = %f, Long =
%f\n",difftime(end, start), FromLatitude, FromLongitude);
                                        time(&start2);
                                }

                                if(ToLongitude >= FromLongitude )
                                {
                                glutIdleFunc(NULL);
                                        k =1;
                                        l = 1;
                                        printf("Time = %f, Steermode LINE %f, %f\n",difftime(end,
start), FromLatitude, FromLongitude);

                                }
                                glutPostRedisplay();
                        }

                }

                /*AUV moving Vertically when FromLatitude is lesser than ToLatitude */

                if (FromLatitude < ToLatitude && j == 0 )
                {

                                FromLatitude = FromLatitude + VMR;
                                MastMotion = 0.008*FromLatitude;

                                if(difftime(end, start3) == 2)
                                {
                                        printf("Time = %f Steermode LINE Lat:%f Long = %f\n",
difftime(end, start), FromLatitude, FromLongitude);
                                        time(&start3);
                                }

                                if((FromLatitude >= ToLatitude) && (FromLongitude ==
ToLongitude))
                                {
```

```
                                        glutIdleFunc(NULL);
                                        j = 1;
                                        printf("Time = %f, Steermode LINE %f, %f\n",difftime(end,
start), FromLatitude, FromLongitude);
                                }
                                else if (FromLatitude>= ToLatitude)
                                {
                                        j = 1;
                                        printf("Stop Lat Time = %f, Steermode LINE %f,
%f\n",difftime(end, start), FromLatitude, FromLongitude);
                                }
                                glutPostRedisplay();
                        }


                /*AUV moving Vertically when FromLatitude is greater than ToLatitude */

                if (FromLatitude > ToLatitude && j == 0 )
                {
                        FromLatitude = FromLatitude - VMR;
                        MastMotion = 0.008*FromLatitude;

                        if (difftime(end, start4) == 2)
                        {
                                printf("Time = %f Steermode LINE Lat:%f Long = %f\n",difftime(end,
start), FromLatitude, FromLongitude);
                                time(&start4);

                        }
                        if((FromLatitude <= ToLatitude) && (FromLongitude == ToLongitude))
                        {
                                glutIdleFunc(NULL);
                                j = 1;
                                printf("Time = %f, Steermode LINE %f, %f\n",difftime(end, start),
FromLatitude, FromLongitude);
                        }
                        else if (FromLatitude <= ToLatitude)
                        {
                                j = 1;
                                printf("Stop Lat motion Time = %f, Steermode LINE %f,
%f\n",difftime(end, start), FromLatitude, FromLongitude);
                        }
                        glutPostRedisplay();

                }

        }

        if(k == 1 && j == 1)
        {
                if(count == 0)
                {
                        DCGOutput = fopen( "C:\\Research\\Animation\\Temp\\Mission\\Position.txt",
"w" );
                        fprintf( DCGOutput, "%f\n", FromLatitude );
                        fprintf( DCGOutput, "%f\n", FromLongitude );
```

```
                                fclose(DCGOutput);
                                DCGOut = fopen( "C:\\Research\\Animation\\Temp\\Mission\\PauseOutput.txt",
"w" );
                                fprintf( DCGOut, "%f\n", FromLatitude );
                                fprintf( DCGOut, "%f\n", FromLongitude );
                                fprintf(DCGOut, "%f\n", difftime(end, start));
                                fclose(DCGOut);
                                DCGTOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt",
"r");
                                fscanf(DCGTOutput,"%f\n", &TotalTime);
                                fclose(DCGTOutput);
                                TotalTime = TotalTime + difftime(end, start);
                                DCGTOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt",
"w");
                                fprintf(DCGTOutput, "%f\n", TotalTime);
                                fclose(DCGTOutput);
                                count = 1;
                                time(&time1);
                                do
                                {
                                        time(&time2);
                                }while(difftime(time2, time1) <5);
                                        printf("Wait for 5 seconds\n");
                                exit(0);
                        }

                }
                time(&end);
}

/*Projection module*/

void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(70.0, (GLfloat) w/(GLfloat) h, 10.0, 4.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef (0.0, 0.0, -5.0);
}

/*Keyboard function*/

void keyboard (unsigned char key, int x, int y)
{
  switch (key) {

     case 'a':
                        {
                                i = 1;
                                  glutIdleFunc(NULL);
                                  printf("MISSION ABORTED\n");
                        }
                       break;
```

```
    case 27:
                        {

      exit(0);
                        }
                        break;
    default:
                        break;
  }
}

/*Mouse action*/

void mouse(int button, int state, int x, int y)

{
        switch(button)
                {
                        case GLUT_LEFT_BUTTON:
                                if (state == GLUT_DOWN && i == 0.0)
                                {
                                        glutIdleFunc(Steer);

                                }
                                break;

                }
}
FILE *DCGInput, *DCGOutput, *DCGIn;
int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        /*Open file for reading input*/
        DCGInput = fopen("C:\\Research\\Animation\\Temp\\DC_Gen_GoToSurface\\DCGInput.txt",
"r");
        if( DCGInput == NULL )
    printf( "The file 'DCGInput.txt' was not opened\n" );// File failed to open
        else
    printf( "The file 'DCGInput.txt' was opened\n" );// File opened

        fseek( DCGInput, 0L, SEEK_SET );
        fscanf(DCGInput, "%f\n", &Timed);//Check whether Timed operation or not (0 (untimed) or 1
(Timed) )
        fscanf(DCGInput, "%f\n", &TimeOfOperation);//Time of operation
        fscanf(DCGInput, "%d\n", &Steering);//Get the order is Steering or not
        fscanf(DCGInput, "%d\n", &SteerMode);//Get the mode of Steering
        //fscanf(SInput, "%f\n", &ToLatitude);//Get the destined Latitude
        fscanf(DCGInput, "%f\n", &ToLongitude);//Get the destned Longitude
   fclose(DCGInput);
        if (Timed == 0)
        {
                VMR = 0.001;
                HMR = 0.001;
        }
        DCGIn = fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "r");
```

```
            fseek( DCGIn, 0L, SEEK_SET );
            fscanf(DCGIn, "%f\n", &FromLatitude);
            fscanf(DCGIn, "%f\n", &FromLongitude);
            fclose(DCGIn);
            printf("Lat = %f, Long = %f\n",FromLatitude, FromLongitude);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMouseFunc(mouse);
    DCGOutput = fopen( "C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "w" );
  fprintf( DCGOutput, "%f\n", ToLatitude );
  fprintf( DCGOutput, "%f\n", ToLongitude );
  fclose( DCGOutput );
  glutMainLoop();
  return 0;
}
```

PayloadDelivery module

```
/*Payload module execution sequence animation*/

#include <GL/glut.h>
#include <stdlib.h>
#include<stdio.h>
#include<time.h>
#include<math.h>
#define VMR 0.0005// Vertical motion rate
#define HMR 0.0005// Horizontal moition rate
#define SurfaceThreshold 1
#define MaxAngle 90
#define MastRate 0.05
#define UP 1
static GLfloat MastAngle;
static GLfloat FromLatitude; // The begining latitude
static GLfloat FromLongitude; //The beginning longitude
static GLfloat MastMotion = 0.0; // Motion of the mast
static GLfloat Slope; // To get the slope of the line in LINE steeringmode
static GLfloat LatLongDiff; // The constant in the equation of a line for LINE steermode
static GLfloat ToLatitude;
static GLfloat ToLongitude;
int SteerMode;
int z=0;
int Steering, count;
int i = 0 ; //Used for abort signal
int j =0, k =0, n =0, l =1;//j for lat loop, k for long loop, l for LINE loop
int Lat = 0, begin, o = 0, m = 0, DevState__MastState;
float TimeOfOperation, TotalTime;
float temp1, temp2, square;
float temp3, number, ExitTime,squareroot;
double elapsed;
time_t start, start1, start2, start3, start4;
```

```
time_t end, end1, end2, end3, end4;
FILE *STOutput, *SOutput, *SteerOutput, *StAngle;

/*Initialization module*/

void init(void)
{

  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_FLAT);
}

/*Display module*/

void display(void)
{
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f(0.0, 0.4, 0.8);
  glRectf(-8.0, -4.0, 8.0, 2.0);
  glPushMatrix();
  glTranslatef (-1.0, 0.0, 0.0);
  glTranslatef (1.0, 0.0, 0.0);
  glTranslatef(FromLongitude, FromLatitude, 0.0);
  glTranslatef(FromLongitude, FromLatitude, 0.0);
  glPushMatrix();
  glScalef (2.0, 0.4, 1.0);
  glColor3f(0.0, 0.7, 0.4);
  glutSolidCube (1.0);
  glPopMatrix();
  glTranslatef (0.0, 0.1, 0.0);
  glTranslatef(0.08*FromLongitude, MastMotion, 0.0);
  glRotatef(0.0, 0.0, 0.0, 1.0);
  glRotatef(MastAngle, 0.0, 0.0, 1.0);
  glTranslatef(0.5, MastMotion, 0.0);
  glPushMatrix();
  glScalef (0.8, 0.2, 0.5);
  glColor3f(1.0, 1.0, 0.0);
  glutSolidCube(1.0);
  glPopMatrix();
  glPopMatrix();
  glutSwapBuffers();
}

/*Module to Steer the AUV*/
void Steer(void)
{
        if(begin == 0 )
        {
                time(&start);
                time(&start1);
                time(&start2);
                time(&start3);
                time(&start4);
                begin++;
        }
```

```
if (Steering == 1)
 {

 /*AUV moving horizontally when FromLongitude is lesser than ToLongitude*/

        if ((FromLongitude < ToLongitude) && (k == 0))
        {

         /*AUV for LINE steering mode to get slope*/
                if ((SteerMode == 1) && (n == 0))
                {
                        Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
                        LatLongDiff = FromLatitude - Slope*FromLongitude;
                        n++; l--;
                }

                /*AUV for LINE steering mode */

                if (SteerMode == 1 && l ==0)
                {
                        FromLongitude = FromLongitude + HMR;
                        FromLatitude = Slope*FromLongitude + LatLongDiff;
                        MastMotion = 0.008*FromLongitude;
                        j = 1;

                        if(difftime(end,start1) == 2)

                        {
                                printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start1), FromLatitude, FromLongitude);
                                time(&start1);
                        }

                        if(ToLongitude <= FromLongitude)
                        {
                        glutIdleFunc(NULL);
                                k =1;
                                l = 1;
                                printf("Time = %f Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);
                        }
                        glutPostRedisplay();
                }

        }


 /*AUV moving horizontally when FromLongitude is greater than ToLongitude*/

        if ((ToLongitude < FromLongitude ) && (k == 0) )
        {

                /*AUV for LINE steering mode to get slope*/

                if (SteerMode == 1 && n == 0)
                {
```

```
                        Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
                        LatLongDiff = FromLatitude - Slope*FromLongitude;
                        n++; l--;
                }

                /*AUV for LINE steering mode*/

                if (SteerMode == 1 && l == 0 )
                {
                        FromLongitude = FromLongitude - HMR;
                        FromLatitude = Slope*FromLongitude + LatLongDiff;
                        MastMotion = 0.008*FromLongitude;
                        j = 1;

                        if((difftime(end,start2)) == 2)
                        {
                                printf("Time = %f, Steermode LINE Lat = %f, Long =
%f\n",difftime(end,start2), FromLatitude, FromLongitude);
                                time(&start2);
                        }

                        if(ToLongitude >= FromLongitude )
                        {
                        glutIdleFunc(NULL);
                                k =1;
                                l = 1;
                                printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);

                        }
                        glutPostRedisplay();

                }

        }


        /*AUV moving Vertically when FromLatitude is lesser than ToLatitude */

        if (FromLatitude < ToLatitude && j == 0 )
        {
                        FromLatitude = FromLatitude + VMR;
                        MastMotion = 0.008*FromLatitude;

                        if(difftime(end,start3) == 2)
                        {
                                if (SteerMode == 1)
                                        printf("Time = %f Steermode LINE Lat:%f Long =
%f\n", difftime(end,start3), FromLatitude, FromLongitude);
                                time(&start3);
                        }

                        if((FromLatitude >= ToLatitude) && (FromLongitude ==
ToLongitude))
                        {
```

```
                                        glutIdleFunc(NULL);
                                        j = 1;
                                        k = 1;
                                        printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);
                                }

                                glutPostRedisplay();

                        }


                /*AUV moving Vertically when FromLatitude is greater than ToLatitude */

                if (FromLatitude > ToLatitude && j == 0 )
                {

                        FromLatitude = FromLatitude - VMR;
                        MastMotion = 0.008*FromLatitude;

                        if(difftime(end,start4)== 2)
                        {
                                if (SteerMode == 1)
                                        printf("Time = %f Steermode LINE Lat:%f Long =
%f\n",difftime(end,start4), FromLatitude, FromLongitude);
                                time(&start4);;

                        }

                        if((FromLatitude <= ToLatitude) && (FromLongitude == ToLongitude))
                        {
                                glutIdleFunc(NULL);
                                j = 1;
                                k = 1;
                                printf("Time = %f, Steermode LINE %f, %f\n",difftime(end,start),
FromLatitude, FromLongitude);
                                }

                                glutPostRedisplay();
                }

        }

        time(&end);
         if(k == 1 && j == 1)
        {
                if(count == 0)
                {
                        SOutput = fopen( "C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "w"
);
                        fprintf( SOutput, "%f\n", FromLatitude );
                        fprintf( SOutput, "%f\n", FromLongitude );
                        fclose(SOutput);

                        SteerOutput = fopen(
"C:\\Research\\Animation\\Temp\\Mission\\PayloadOutput.txt", "w" );
```

```
                        fprintf(SteerOutput, "Payload delivered at the given location\n");
                        fprintf( SteerOutput, "%f\n", FromLatitude );
                        fprintf( SteerOutput, "%f\n", FromLongitude );
                        fprintf(SteerOutput, "%f\n", difftime(end, start));
                        fclose(SteerOutput);
                        STOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "r");
                        fscanf(STOutput,"%f\n", &TotalTime);
                        fclose(STOutput);
                        TotalTime = TotalTime + difftime(end, start);
                        STOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "w");
                        fprintf(STOutput, "%f\n", TotalTime);
                        fclose(STOutput);
                        count = 1;
                        exit(0);
                }
        }
}

void Abort(void)
{
/*AUV moving up to take GPSFix (Control within GPSFix module)*/
        /*Take value of SurfaceThreshold from Uppaal file */
        if (FromLatitude < SurfaceThreshold && o ==0)
        {
                FromLatitude = FromLatitude + VMR;
                MastMotion = 0.008*FromLatitude;

                if(FromLatitude == SurfaceThreshold)
                {
                glutIdleFunc(NULL);
                        o = 1;
                }
                glutPostRedisplay();
        }

        if (FromLatitude >= SurfaceThreshold && m == 0)
        {
                MastAngle = MastAngle + MastRate;
                /*Rate to raise mast when AUV is on water surface rate valur from Uppaaal*/
                while (MastAngle>MaxAngle)  /*Take value of MastAngle from Uppaal*/
                {
                        DevState__MastState = UP;
                        m = 1;
                }

                glutPostRedisplay();
        }

}


/*Projection module*/

void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
```

```
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(70.0, (GLfloat) w/(GLfloat) h, 10.0, 4.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef (0.0, 0.0, -5.0);
}

/*Keyboard function*/

void keyboard (unsigned char key, int x, int y)
{
  switch (key) {

    case 'a':
                        {
                                i = 1;
                                glutIdleFunc(Abort);
                                printf("MISSION ABORTED\n");
                        }
                        break;

    case 27:
                        {

      exit(0);
                        }
                        break;
    default:
                        break;
  }
}

/*Mouse action*/

void mouse(int button, int state, int x, int y)

{

        switch(button)
                {

                        case GLUT_LEFT_BUTTON:

                                if (state == GLUT_DOWN && i == 0.0)
                                {

                                        glutIdleFunc(Steer);

                                }
                                break;

                }
}

FILE *SInput, *StInput, *SAngle;
```

```
int main(int argc, char** argv)
{
	glutInit(&argc, argv);

	/*Open file for reading input*/

	SInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\SteerInput.txt", "r");
	if( SInput == NULL )
    printf( "The file 'SteerInput.txt' was not opened\n" );// File failed to open
	else
    printf( "The file 'SteerInput.txt' was opened\n" );// File opened

	fseek( SInput, 0L, SEEK_SET );
	fscanf(SInput, "%d\n", &Steering);//Get the order is Steering or not
	fscanf(SInput, "%d\n", &SteerMode);//Mode of Steering
	fscanf(SInput, "%f\n", &ToLatitude);//Get the destined Latitude
	fscanf(SInput, "%f\n", &ToLongitude);//Get the destned Longitude
  fclose(SInput);
	printf("ToLat = %f, ToLong = %f\n",ToLatitude, ToLongitude);

	if (ToLatitude < -1.5)
	{
		ToLatitude = -1.5;
		printf("ToLatitude value change to -1.5 as depth below that is dangerous\n");
	}
	if (ToLatitude > 1.0)
	{
		ToLatitude = 1.0;
		printf("ToLatitude value cannot be more than 1.0 the surface\n");
	}

	StInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "r");
	fscanf(SInput, "%f\n", &FromLatitude);//Get the current Latitude
	fscanf(SInput, "%f\n", &FromLongitude);//Get the current Longitude
	printf("Lat = %f, Long = %f\n",FromLatitude, FromLongitude);

	if (FromLatitude < -1.5)
	{
		FromLatitude = -1.5;
		printf("FromLatitude value change to -1.5 as depth below that is dangerous\n");
	}

	if (FromLatitude > 1.0)
	{
		FromLatitude = 1.0;
		printf("FromLatitude value cannot be more than 1.0 the surface\n");
	}

	temp1 = FromLatitude - ToLatitude;
	temp2 = FromLongitude - ToLongitude;
	temp3 = temp1*temp1 + temp2*temp2;
	squareroot = sqrt(temp3);
	printf("%f\n", squareroot);

	SAngle = fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "r");
	fscanf(SAngle, "%f\n", &MastAngle);
```

```
        if (MastAngle > 0 && FromLatitude < 1.0)
        {
                MastAngle = 0.0;
                printf("Mast is not raised as not on surface\n");
        }
        fclose(SAngle);
        if (MastAngle > 0 && FromLatitude == 1.0)
        {
                MastAngle = 0.0;
                printf("Lower Mast before going below surface of water\n");
        }

  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMouseFunc(mouse);
  glutMainLoop();
  return 0;
}
```

Launch

```
/*Loiter module execution sequence animation*/

#include <GL/glut.h>
#include <stdlib.h>
#include<stdio.h>
#include<math.h>
#include <time.h>
#define VMR 0.001//Rate to move up depeding
#define HMR 0.001 //Rate to move horizontally
#define StartPoint -1.0 //Keeping a record of start point
#define Steering 1//Checks whether steering mode received or not
#define Direct 1 //Steering mode direct to go straight to the point following a line
#define HOVER 1
#define CIRCLE 2
#define NONE 0
#define PI 3.14
#define Steering 1
#define LoiterTO 10000
#define Loiter_TimeToLoiterPt_FCN 10
#define ToWP__Timed 1 // Timed waypoint if 1
#define TimeToWaypoint 10000
#define MinSpeed 0.001
#define ThresholdDistance 20
#define MastRate 0.02
#define DOWN 1
static GLfloat MastAngle = 90.0;
static GLfloat DevState__MastState = 0;
static GLfloat Helm__DistanceToPoint;
static GLfloat FromLatitude = 1.0; // The begining latitude
```

```
static GLfloat FromLongitude = -1.0; //The beginning longitude
static GLfloat MastMotion = 0.0; // Motion of the mast
static GLfloat i = 0; //Used for abort signal
static GLfloat Slope; // To get the slope of the line in direct steeringmode
static GLfloat SteerMode = Direct;//Assigning mode to steering
static GLfloat LatLongDiff; // The constant in the equation of a line for direct steermode
static GLfloat Step;
static GLfloat ToWP__LoiterType = 2;
static GLfloat Loiter = 0;
static GLfloat x, y;
static GLfloat First = 0;
static GLfloat r;
static GLfloat Theta;
static GLfloat base;
static LoiterDone = 0;
static GLfloat m = 0;
static GLfloat Side, Long =0;
static GLfloat TimeInState;
static GLfloat Helm =0;
static GLfloat DistanceToWaypoint;
static GLfloat t;
static GLfloat ToLatitude = 0.5;
static GLfloat ToLongitude = 1.0;
static GLfloat ToWP__LoiterDuration = 5;
static GLfloat ActualTimeToWP;
static GLfloat RemainingTime;
static GLfloat DecideLoiter = 0;
int ToWP__LoiterAtWP = 1;
int WaypointNavigator_DepthTrouble_FCN;
int MastRaised = 0;
int Initial = 0;
int j =0, k =0, n =0, l =1, p=0;//j for lat loop, k for long loop, l for direct loop, p for indirect
int MastLowered = 0;
int NumPoints = 2, numCount;
int begin = 0;
double LowerMast, TotalTime;
time_t start, start1, start2, start3, start4, start5;
time_t end, end1, end2, end3;
FILE *LaunchPOutput, *LaunchTime, *LaunchAOutput;

void init(void)
{

  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_FLAT);
}

void display(void)
{
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f(0.0, 0.4, 0.8);
  glRectf(-4.0, -4.0, 4.0, 2.0);
  glPushMatrix();
  glTranslatef (-1.0, 0.0, 0.0);
  glTranslatef (1.0, 0.0, 0.0);
  glTranslatef(FromLongitude, FromLatitude, 0.0);
```

```
  glTranslatef(FromLongitude, FromLatitude, 0.0);
  glPushMatrix();
  glScalef (2.0, 0.4, 1.0);
  glColor3f(0.0, 0.7, 0.4);
  glutSolidCube (1.0);
  glPopMatrix();
  glTranslatef (0.0, 0.1, 0.0);
  glTranslatef(0.08*FromLongitude, MastMotion, 0.0);
  glRotatef(MastAngle, 0.0, 0.0, 1.0);
  glTranslatef(0.5, MastMotion, 0.0);
  glPushMatrix();
  glScalef (0.8, 0.2, 0.5);
  glColor3f(1.0, 1.0, 0.0);
  glutSolidCube(1.0);
  glPopMatrix();
  glPopMatrix();
  glutSwapBuffers();
}
void MastLower(void)
{
        if(begin == 0)
        {
                time(&start);
                begin++;
        }

        if(MastAngle <= 0 && numCount == 0 && MastLowered == 0)
        {
                printf("Mast is already lowered\n");
                MastLowered++;
                numCount++;
                begin = 2;
        }
 /***Lower mast when GPSFix is taken (Control passed to Launch module)***/
                if (FromLatitude >= 1 && MastAngle > 0 && MastLowered == 0)
                {
                        if(begin == 1)
                        {
                                time(&start1);
                                begin++;
                        }
                        MastAngle = MastAngle - MastRate;/*Lowering mast rate specified in uppaal*/


                        if (difftime(end1, start1) == 1)
                        {
                                LowerMast = LowerMast + difftime(end1, start1);
                                printf ("MissionTime = %f, LowerMast = %f, MastAngle =
%f\n",difftime(end1, start), LowerMast, MastAngle);
                                time(&start1);
                        }
                                if (MastAngle < 0.0)
                        {
                                DevState__MastState = DOWN;
                                MastLowered = 1;
```

```
                }
            glutPostRedisplay();

            }
            time(&end1);


if (MastLowered == 1)
{

    /***Lower mast when GPSFix is taken (Control passed to Launch module)***/


    if (Steering == 1)
     {
        if(begin == 2)
        {
            time(&start2);
            time(&start3);
            time(&start4);
            time(&start5);
            //printf("Begin = %d\n", begin);
            begin++;
        }
        if((FromLongitude == ToLongitude) && (FromLatitude == ToLatitude))
        {
            k = 1;
            j = 1;
            exit(0);
        }

     /*AUV moving horizontally when FromLongitude is lesser than ToLongitude*/

        if ((FromLongitude < ToLongitude) && (k == 0))
        {


         /*AUV for LINE steering mode to get slope*/


            if ((SteerMode == 1) && (n == 0))
            {
                Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
                LatLongDiff = FromLatitude - Slope*FromLongitude;
                n++; l--;
            }

            /*AUV for LINE steering mode */

            if (SteerMode == 1 && l ==0)
            {
                FromLongitude = FromLongitude + HMR;
                FromLatitude = Slope*FromLongitude + LatLongDiff;
                MastMotion = 0.008*FromLongitude;
                j = 1;
```

```
                                        if(difftime(end2,start2) == 1)

                                        {
                                                printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);
                                                time(&start2);
                                        }

                                        if(ToLongitude <= FromLongitude)
                                        {
                                        glutIdleFunc(NULL);
                                                k =1;
                                                l = 1;
                                                printf("Time = %f Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);
                                        }
                                        glutPostRedisplay();
                                }


                        } time(&end2);

                /*AUV moving horizontally when FromLongitude is greater than ToLongitude*/

                        if ((ToLongitude < FromLongitude ) && (k == 0) )
                        {

                                /*AUV for LINE steering mode to get slope*/

                                if (SteerMode == 1 && n == 0)
                                {
                                        Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
                                        LatLongDiff = FromLatitude - Slope*FromLongitude;
                                        n++; l--;
                                }

                                /*AUV for LINE steering mode*/

                                if (SteerMode == 1 && l == 0 )
                                {
                                        FromLongitude = FromLongitude - HMR;
                                        FromLatitude = Slope*FromLongitude + LatLongDiff;
                                        MastMotion = 0.008*FromLongitude;
                                        j = 1;

                                        if((difftime(end3,start3)) == 1)
                                        {
                                                printf("Time = %f, Steermode LINE Lat = %f, Long =
%f\n",difftime(end,start), FromLatitude, FromLongitude);
                                                time(&start3);
                                        }

                                        if(ToLongitude >= FromLongitude )
                                        {
                                        glutIdleFunc(NULL);
```

```
                                        k =1;
                                        l = 1;
                                        printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);

                                }
                                glutPostRedisplay();

                        }

                } time(&end3);


                /*AUV moving Vertically when FromLatitude is lesser than ToLatitude */



                if (FromLatitude < ToLatitude && j == 0 )
                {
                                FromLatitude = FromLatitude + VMR;
                                MastMotion = 0.008*FromLatitude;

                                if(difftime(end,start4) == 2)
                                {
                                        if (SteerMode == 1)
                                                printf("Time = %f Steermode LINE Lat:%f Long =
%f\n", difftime(end,start), FromLatitude, FromLongitude);
                                        time(&start4);
                                }

                                if((FromLatitude >= ToLatitude) && (FromLongitude ==
ToLongitude))
                                {

                                        glutIdleFunc(NULL);
                                        j = 1;
                                        k = 1;
                                        printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);
                                }

                                glutPostRedisplay();

                }


                /*AUV moving Vertically when FromLatitude is greater than ToLatitude */
                if (FromLatitude > ToLatitude && j == 0 )
                {
                        FromLatitude = FromLatitude - VMR;
                        MastMotion = 0.008*FromLatitude;

                        if(difftime(end,start5)== 2)
                        {
                                if (SteerMode == 1)
```

```
                                                printf("Time = %f Steermode LINE Lat:%f Long =
%f\n",difftime(end,start), FromLatitude, FromLongitude);
                                        time(&start5);;

                                }

                                if((FromLatitude <= ToLatitude) && (FromLongitude == ToLongitude))
                                {
                                        glutIdleFunc(NULL);
                                        j = 1;
                                        k = 1;
                                        printf("Time = %f, Steermode LINE %f, %f\n",difftime(end,start),
FromLatitude, FromLongitude);
                                }

                                glutPostRedisplay();


                        }


                }

                time(&end);
               }
               if (k == 1 && j == 1)
                        {

                                        LaunchAOutput =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Angle.txt", "w" );
                                        fprintf( LaunchAOutput, "%f\n", MastAngle );
                                        fclose( LaunchAOutput );

                                        LaunchPOutput =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "w" );
                                        fprintf( LaunchPOutput, "%f\n", FromLatitude);
                                        fprintf( LaunchPOutput, "%f\n", FromLongitude);
                                        fclose( LaunchPOutput );

                                        LaunchTime =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "r" );
                                        fseek( LaunchTime, 0L, SEEK_SET );
                                        fscanf(LaunchTime, "%f\n", &TotalTime);
                                        fclose(LaunchTime);
                                        printf("TotalTime = %f\n", TotalTime);
                                        TotalTime = TotalTime + difftime(end, start);


                                        LaunchTime =
fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "w" );
                                        fseek( LaunchTime, 0L, SEEK_SET );
                                        fprintf(LaunchTime, "%f\n", TotalTime);
                                        fclose(LaunchTime);
                                        //exit(0);
                                        k++; j++;
                        }
```

```
}
void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(70.0, (GLfloat) w/(GLfloat) h, 10.0, 4.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef (0.0, 0.0, -5.0);
}

void keyboard (unsigned char key, int x, int y)
{
  switch (key) {

    case 'a':
                        {
                                i = 1;
                                  glutIdleFunc(NULL);
                        }
                       break;

    case 27:
      exit(0);
      break;
    default:
      break;
  }
}
void mouse(int button, int state, int x, int y)

{
        switch(button)
                {
                        case GLUT_LEFT_BUTTON:
                                if (state == GLUT_DOWN && i == 0.0)
                                {
                                        glutIdleFunc(MastLower);

                                }
                                break;


                }
}

FILE *LaunchInput, *LaInput;
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  LaunchInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "r");

        if( LaunchInput == NULL )
    printf( "The file 'Position.txt' was not opened\n" );// File failed to open
        else
```

```
    printf( "The file 'Position.txt' was opened\n" );// File opened
        fseek( LaunchInput, 0L, SEEK_SET );
        fscanf(LaunchInput, "%f\n", &FromLatitude);//Get the destined Latitude
        fscanf(LaunchInput, "%f\n", &FromLongitude);//Get the destned Longitude
        printf("FromLat = %f FromLong = %f\n", FromLatitude, FromLongitude);
        fclose(LaunchInput);
        LaInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Angle.txt", "r");
        fseek( LaInput, 0L, SEEK_SET );
        fscanf(LaInput, "%f\n", &MastAngle);
        fclose(LaInput);
        if(MastAngle > 0 && FromLatitude < 1 && MastLowered == 0)
        {
                MastAngle = 0;
                MastLowered = 1;
                printf("Below surface so mast is already lowered\n");
        }

  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMouseFunc(mouse);
  glutMainLoop();
  return 0;
}
```

WaypointNavigator

```
/*Waypoint module execution sequence animation*/

#include <GL/glut.h>
#include <stdlib.h>
#include<stdio.h>
#include<math.h>
#include<time.h>
#define VerticalMotionRate 0.001//Rate to move up depeding
#define HorzMotionRate 0.001 //Rate to move horizontally
#define DriftRate 0.001 //To assign a value to water current given as drift
#define StartPoint -1.0 //Keeping a record of start point
#define Steering 1//Checks whether steering mode received or not
#define Direct 1 //Steering mode direct to go straight to the point following a line
#define NotDirect 2 //Steering mode indirect
#define HOVER 1
#define CIRCLE 2
#define NONE 0
#define PI 3.14
#define Steering 1
#define LoiterTO 50
#define Loiter_TimeToLoiterPt_FCN 25
#define ToWP__Timed 0 // Timed waypoint if 1
#define TimeToWaypoint 25
#define Speed 0.35
```

```
#define ThresholdDistance 0.1
#define HMR 0.001
#define VMR 0.001
static GLfloat Helm__DistanceToPoint;
static GLfloat FromLatitude ; // The begining latitude
static GLfloat FromLongitude ; //The beginning longitude
static GLfloat MastMotion = 0.0; // Motion of the mast
static GLfloat i = 0; //Used for abort signal
static GLfloat Slope; // To get the slope of the line in direct steeringmode
static GLfloat SteerMode = Direct;//Assigning mode to steering
static GLfloat LatLongDiff; // The constant in the equation of a line for direct steermode
static GLfloat Step;
static GLfloat ToWP__LoiterType = 1;
static GLfloat Loiter = 0;
static GLfloat x, y;
static GLfloat First = 0;
static GLfloat r;
static GLfloat Theta;
static GLfloat base;
static LoiterDone = 0;
static GLfloat MTime = 0;
static GLfloat m = 0;
static GLfloat Side, Long =0;
static GLfloat TimeInState;
static GLfloat Helm =0;
static GLfloat DistanceToWaypoint;
static GLfloat MastAngle;
static GLfloat ToLatitude ;
static GLfloat ToLongitude ;
static GLfloat ToWP__LoiterDuration = 5;
static GLfloat ActualTimeToWP;
static GLfloat RemainingTime;
static GLfloat DecideLoiter = 0;
static GLfloat TimeInLoiter = 0;
static GLfloat HoverPoint;
int j =0, k =0, n =0, l =1, p=0;//j for lat loop, k for long loop, l for direct loop, p for indirect
int NumPoints = 2;
int Initial = 0, intCount;
int WaypointNavigator_DepthTrouble_FCN;
int ToWP__LoiterAtWP = 1;
int begin = 0, Loitering = 0, count = 0;
int Hove = 0;
int iteration = 0;
float TotalTime;
time_t start, start1, start2, start3, start4, start5, startn, startm;
time_t end, end1, end2, end3, endn, endn1, startn1;
FILE *WOutput, *WptOutput, *WTOutput;
void init(void)
{

  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_FLAT);
}

void display(void)
{
```

```
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f(0.0, 0.4, 0.8);
  glRectf(8.0, -4.0, -8.0, 2.0);
  glPushMatrix();
  glTranslatef (-1.0, 0.0, 0.0);
  glTranslatef (1.0, 0.0, 0.0);
  glTranslatef(FromLongitude, FromLatitude, 0.0);
  glTranslatef(FromLongitude, FromLatitude, 0.0);
  glPushMatrix();
  glScalef (2.0, 0.4, 1.0);
  glColor3f(0.0, 0.7, 0.4);
  glutSolidCube (1.0);
  glPopMatrix();
  glTranslatef (0.0, 0.1, 0.0);
  glTranslatef(0.08*FromLongitude, MastMotion, 0.0);
  glRotatef(0.0, 0.0, 0.0, 1.0);
  glTranslatef(0.5, MastMotion, 0.0);
  glPushMatrix();
  glScalef (0.8, 0.2, 0.5);
  glColor3f(1.0, 1.0, 0.0);
  glutSolidCube(1.0);
  glPopMatrix();
  glPopMatrix();
  glutSwapBuffers();
}

void MoveUp(void)
{
        if (Initial == 0)
        {
                DistanceToWaypoint = pow(pow((FromLatitude-ToLatitude),2) + pow((FromLongitude-
ToLongitude),2), 0.5);
                ActualTimeToWP =  DistanceToWaypoint/Speed;
                RemainingTime = TimeToWaypoint-ActualTimeToWP;
                Initial = 1;
                printf("DTW = %f ActualTimeToWP = %f\n",DistanceToWaypoint, ActualTimeToWP);
        }

        if (ToWP__Timed == 1 && (TimeToWaypoint>ActualTimeToWP) && (TimeToWaypoint-
difftime(endn1, startn) >=  ActualTimeToWP) )
        {

                if(begin == 0)
                {
                        time(&startn);
                        time(&startn1);
                        begin++;
                }
                if (ToLatitude <= -2.0 )
                {
                        WaypointNavigator_DepthTrouble_FCN = 1;
                }

                if (WaypointNavigator_DepthTrouble_FCN == 1)
                {
                        ToLatitude = - 1.8;
```

```
                        printf("Depth trouble corrected\n");
                }

                STEER();
                if(difftime(endn1, startn1) == 2)
                {
                        printf("Time = %f, FromLatitude = %f, FromLongitude\n", difftime(endn1,
startn), FromLatitude, FromLongitude);
                        time(&startn1);
                }
        }
                time(&endn1);


                if (ToWP__Timed == 1 && ToWP__LoiterDuration > 0 && (TimeToWaypoint-
difftime(endn1, startn) <=  ActualTimeToWP))
                {

                        if(begin == 1)
                        {
                                printf("Time to reach WP = %f, Lat = %f, Long = %f\n",
difftime(endn1, startn), FromLatitude, FromLongitude);
                                time(&start);
                                time(&start1);
                                begin++;
                        }

                        if ((NumPoints<2 && LoiterTO == 0 || ToWP__LoiterType == NONE) &&
LoiterDone == 0)
                        {
                                LoiterDone = 1;
                                glutIdleFunc(NULL);
                                printf("No time to Loiter \n");
                        }

                        if ((NumPoints<2 && LoiterTO>0) || ToWP__LoiterType == HOVER &&
LoiterDone == 0)
                        {

                                if (count == 0)
                                {
                                        HoverPoint = FromLongitude;
                                        printf("HOVERING, HoverPoint = %f, Hove \n",
HoverPoint);
                                        count++;
                                }

                        if (FromLongitude < (HoverPoint + 1.0) && Hove == 0 && LoiterDone == 0)
                        {
                                FromLongitude = FromLongitude + 0.001;

                                if (FromLongitude >= HoverPoint + 1.0)
                                {
                                        ++Hove;
                                        printf("Hove in fwd= %d, Time Hovering = %f\n", Hove,
difftime(end1, start));
```

```
                                        }
                                        if ((difftime(end1, start) >= (LoiterTO-Loiter_TimeToLoiterPt_FCN-
20)))
                                        {
                                                LoiterDone = 1;

                                                printf("LoiterDone\n");
                                        }
                                }

                                if (FromLongitude > HoverPoint  && Hove == 1 && LoiterDone == 0)
                                {
                                        FromLongitude = FromLongitude - 0.001;
                                        if (FromLongitude <= HoverPoint)
                                        {
                                                --Hove;
                                                printf("Hove in bwd= %d Time Hovering = %f \n",Hove,
difftime(end1, start));
                                        }
                                        if ((difftime(end1, start) >= (LoiterTO-Loiter_TimeToLoiterPt_FCN-
20)))
                                        {
                                                LoiterDone = 1;

                                        }
                                }

                                glutPostRedisplay();
                        }
                        time(&end1);

                        if (NumPoints>=2 && LoiterTO>0 && ToWP__LoiterType == CIRCLE &&
(difftime(end2, start)<=LoiterTO-Loiter_TimeToLoiterPt_FCN) && Helm ==0)
                        {
                                Step = 360/NumPoints;
                                for (m=0; m<Step; m=m+1)
                                {
                                        if (First == 0)
                                        {
                                                First = 1;
                                                r = pow(pow((FromLatitude-ToLatitude),2) +
pow((FromLongitude-ToLongitude),2), 0.5);
                                                Helm__DistanceToPoint = r;
                                                if (Helm__DistanceToPoint <= 0.25)
                                                {
                                                        glutIdleFunc(NULL);
                                                        printf("Distance to point is less than 20 m\n");
                                                        Helm++;
                                                }
                                                Side = pow(pow((FromLatitude-ToLatitude-r),2) +
pow((FromLongitude-ToLongitude),2), 0.5);
                                                Theta = 2*asin(Side/(2*r));
                                                printf("r=%f, Theta = %f\n",r, Theta);
                                                FromLongitude = -r * cos(Theta) + ToLongitude;
                                                FromLatitude = -r * sin(Theta) + ToLatitude;
                                                glutPostRedisplay();
```

```
                                        m++;
                                        printf("FromLat = %f, FromLong = %f, m = %f\n",
FromLatitude, FromLongitude, m);
                                }

                                if (m>0 && LoiterDone == 0)
                                {
                                        Theta = Theta + 0.00001;
                                        FromLongitude = -r * cos(Theta)+ ToLongitude;
                                        FromLatitude = -r * sin(Theta)+ ToLatitude;

                                        if (difftime(end2, start1) == 2)
                                        {
                                                printf("TimeInState = %f, FromLat = %f, FromLong
= %f\n", difftime(end2, start), FromLatitude, FromLongitude);
                                                time(&start1);
                                        }

                                        if (difftime(end2, start)>=LoiterTO-
Loiter_TimeToLoiterPt_FCN)
                                        {
                                                LoiterDone = 1;
                                        }

                                        if(FromLatitude > 1.0)
                                        {
                                                FromLatitude = 1;
                                        }
                                        if(FromLatitude < -1.5)
                                        {
                                                FromLatitude = -1.5;
                                        }

                                        glutPostRedisplay();
                                }
                        }

                }

                time(&end2);

                if (ToWP__LoiterDuration <= 0)
                {
                        LoiterDone = 1;
                }

                if (difftime(end2, startn) >= TimeToWaypoint && LoiterDone ==1 && iteration == 0)
                {
                        printf("Time for WP = %f\n",difftime(end2, startn));
                        printf("No time to  reach waypoint\n");
                        iteration++;

                }

                if (ToWP__LoiterAtWP == 1 && (difftime(end2, startn) <= TimeToWaypoint) &&
LoiterDone == 1)
```

```
		{
			Helm__DistanceToPoint = pow(pow((FromLatitude-ToLatitude),2) +
pow((FromLongitude-ToLongitude),2), 0.5);

			//printf("Helm = %f\n",Helm__DistanceToPoint);
			if ((ToWP__Timed == 0 && Helm__DistanceToPoint <= ThresholdDistance)
||(ToWP__Timed == 1 && Helm__DistanceToPoint<=5))
			{
				if(begin == 2)
				{

					time(&startm);
					time(&start5);
					printf("Helm = %f\n",Helm__DistanceToPoint);
					Slope =(ToLatitude-FromLatitude)/(ToLongitude-
FromLongitude);
					LatLongDiff = FromLatitude - Slope*FromLongitude;
					begin++;
				}

				if (FromLongitude > ToLongitude && k == 0)
				{
					FromLongitude = FromLongitude - 0.001;
					FromLatitude = Slope*FromLongitude + LatLongDiff;
					MastMotion = 0.008*FromLongitude;
					j = 1;
					if(difftime(endn,start5) == 2)

					{
						printf("Hey Time = %f, Steermode LINE %f,
%f\n",difftime(endn,start5), FromLatitude, FromLongitude);
						time(&start5);
					}
					if(ToLongitude >= FromLongitude)
					{
					glutIdleFunc(NULL);
						k =1;
						l = 1;
						printf("Hello Time = %f Steermode LINE %f,
%f\n",difftime(endn,startm), FromLatitude, FromLongitude);
					}
					glutPostRedisplay();
				}


				if(FromLongitude < ToLongitude)
				{
					FromLongitude = FromLongitude + 0.001;
					FromLatitude = Slope*FromLongitude + LatLongDiff;
					MastMotion = 0.008*FromLongitude;
					j = 1;

					if(difftime(endn,start5) == 2)

					{
```

```
                                        printf("Time = %f, Steermode LINE %f,
%f\n",difftime(endn,start5), FromLatitude, FromLongitude);
                                        time(&start5);
                                }
                                if(ToLongitude <= FromLongitude)
                                {
                                glutIdleFunc(NULL);
                                        k =1;
                                        l = 1;
                                        printf("LoiterDone = %d\n", LoiterDone);
                                        printf("Time = %f Steermode LINE %f,
%f\n",difftime(endn,startm), FromLatitude, FromLongitude);
                                }
                                glutPostRedisplay();
                        }

                        if(FromLongitude == ToLongitude && k == 0)
                        {
                                if(FromLatitude < ToLatitude && j == 0)
                                {
                                        FromLatitude = FromLatitude + VMR;
                                        MastMotion = 0.008*FromLatitude;

                                        if(difftime(endn,start5) == 2)
                                        {
                                                printf("Time = %f Steermode LINE Lat:%f
Long = %f\n", difftime(endn,start5), FromLatitude, FromLongitude);
                                                time(&start5);
                                        }

                                        if(FromLatitude >= ToLatitude)

                                        {
                                                glutIdleFunc(NULL);
                                                j = 1;
                                                k = 1;
                                                printf("Time = %f, Steermode LINE %f,
%f\n",difftime(endn,startm), FromLatitude, FromLongitude);
                                        }

                                        glutPostRedisplay();
                                }

                                if(FromLatitude > ToLatitude && j == 0)
                                {
                                        FromLatitude = FromLatitude + VMR;
                                        MastMotion = 0.008*FromLatitude;

                                        if(difftime(endn,start5) == 2)
                                        {
                                                printf("Time = %f Steermode LINE Lat:%f
Long = %f\n", difftime(endn,start5), FromLatitude, FromLongitude);
                                                time(&start5);
                                        }
```

```
                                                        if(FromLatitude <= ToLatitude)

                                                        {
                                                                glutIdleFunc(NULL);
                                                                j = 1;
                                                                k = 1;
                                                                printf("Time = %f, Steermode LINE %f,
%f\n",difftime(endn,start5), FromLatitude, FromLongitude);
                                                        }

                                                        glutPostRedisplay();
                                                }
                                        }


                                }
                        }

                }
                time(&endn);

                if(k == 1 && j == 1 && LoiterDone == 1 )
                 {

                        if(intCount == 0)
                        {
                                printf("Jai he\n");
                                WOutput = fopen( "C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "w"
);
                                fprintf( WOutput, "%f\n", FromLatitude );
                                fprintf( WOutput, "%f\n", FromLongitude );
                                fclose(WOutput);

                                WptOutput = fopen(
"C:\\Research\\Animation\\Temp\\Mission\\WptOutput.txt", "w" );
                                fprintf( WptOutput, "%f\n", FromLatitude );
                                fprintf( WptOutput, "%f\n", FromLongitude );
                                fprintf(WptOutput, "%f\n", difftime(end, start));
                                fclose(WptOutput);

                                WTOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "r");
                                fscanf(WTOutput,"%f\n", &TotalTime);
                                fclose(WTOutput);

                                TotalTime = TotalTime + difftime(endn, startn);

                                WTOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "w");
                                fprintf(WTOutput, "%f\n", TotalTime);
                                fclose(WTOutput);
                                intCount++;
                                exit(0);
                        }
                 }

                if (ToWP__Timed == 0 || TimeToWaypoint <= DistanceToWaypoint/Speed)
                {
```

```
		if(begin == 0)
		{
			time(&startn);
			begin++;
		}

		STEER();

		if (ToLongitude <= -2.0 )
		{
			WaypointNavigator_DepthTrouble_FCN = 1;
		}

		if (WaypointNavigator_DepthTrouble_FCN ==1)
		{
			while(ToLongitude < -2.0)
			{
				ToLongitude = ToLongitude - 0.2;
			}
			printf("Depth trouble corrected\n");
		}

		Helm__DistanceToPoint = pow(pow((FromLatitude-ToLatitude),2) +
pow((FromLongitude-ToLongitude),2), 0.5);
		if ((ToWP__Timed == 0 &&
Helm__DistanceToPoint<=ThresholdDistance)||(ToWP__Timed == 1
&&Helm__DistanceToPoint<=0.55))
		{

			glutIdleFunc(NULL);
			printf("Time = %f\n", difftime(endn, startn));
			WOutput = fopen( "C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "w"
);
			fprintf( WOutput, "%f\n", FromLatitude );
			fprintf( WOutput, "%f\n", FromLongitude );
			fclose(WOutput);

			WptOutput = fopen(
"C:\\Research\\Animation\\Temp\\Mission\\WptOutput.txt", "w" );
			fprintf( WptOutput, "%f\n", FromLatitude );
			fprintf( WptOutput, "%f\n", FromLongitude );
			fprintf(WptOutput, "%f\n", difftime(endn, startn));
			fclose(WptOutput);

			WTOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "r");
			fscanf(WTOutput,"%f\n", &TotalTime);
			fclose(WTOutput);

			TotalTime = TotalTime + difftime(endn, startn);

			WTOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "w");
			fprintf(WTOutput, "%f\n", TotalTime);
			fclose(WTOutput);
			count = 1;
			exit(0);
```

```
                }
        }
        time(&endn);

        if(k == 1 && j == 1 || LoiterDone == 1)
         {
                if(count == 0)
                {
                        WOutput = fopen( "C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "w"
);
                        fprintf( WOutput, "%f\n", FromLatitude );
                        fprintf( WOutput, "%f\n", FromLongitude );
                        fclose(WOutput);

                        WptOutput = fopen(
"C:\\Research\\Animation\\Temp\\Mission\\WptOutput.txt", "w" );
                        fprintf( WptOutput, "%f\n", FromLatitude );
                        fprintf( WptOutput, "%f\n", FromLongitude );
                        fprintf(WptOutput, "%f\n", difftime(end, start));
                        fclose(WptOutput);

                        WTOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "r");
                        fscanf(WTOutput,"%f\n", &TotalTime);
                        fclose(WTOutput);

                        TotalTime = TotalTime + difftime(endn, start);

                        WTOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "w");
                        fprintf(WTOutput, "%f\n", TotalTime);
                        fclose(WTOutput);
                        count = 1;
                        exit(0);
                }
         }
}

STEER()
{
                if(begin == 0 )
        {
                time(&start);
                time(&start1);
                time(&start2);
                time(&start3);
                time(&start4);
                begin++;
        }

        if (Steering == 1 && Helm == 0)
         {

         /*AUV moving horizontally when FromLongitude is lesser than ToLongitude*/

                if ((FromLongitude < ToLongitude) && (k == 0))
                {
```

```
/*AUV for LINE steering mode to get slope*/


        if ((SteerMode == 1) && (n == 0))
        {
                Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
                LatLongDiff = FromLatitude - Slope*FromLongitude;
                n++; l--;
        }

        /*AUV for LINE steering mode */

        if (SteerMode == 1 && l ==0)
        {
                FromLongitude = FromLongitude + HMR;
                FromLatitude = Slope*FromLongitude + LatLongDiff;
                MastMotion = 0.008*FromLongitude;
                j = 1;

                if(difftime(end,start1) == 2)

                {
                        printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start1), FromLatitude, FromLongitude);
                        time(&start1);
                }

                if(ToLongitude <= FromLongitude)
                {
                glutIdleFunc(NULL);
                        k =1;
                        l = 1;
                        printf("Time = %f Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);
                }
                glutPostRedisplay();
        }


}


/*AUV moving horizontally when FromLongitude is greater than ToLongitude*/

        if ((ToLongitude < FromLongitude ) && (k == 0) )
        {

                /*AUV for LINE steering mode to get slope*/

                if (SteerMode == 1 && n == 0)
                {
                        Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
                        LatLongDiff = FromLatitude - Slope*FromLongitude;
                        n++; l--;
                }
```

```
                        /*AUV for LINE steering mode*/

                        if (SteerMode == 1 && l == 0 )
                        {
                                FromLongitude = FromLongitude - HMR;
                                FromLatitude = Slope*FromLongitude + LatLongDiff;
                                MastMotion = 0.008*FromLongitude;
                                j = 1;

                                if((difftime(end,start2)) == 2)
                                {
                                        printf("Time = %f, Steermode LINE Lat = %f, Long =
%f\n",difftime(end,start2), FromLatitude, FromLongitude);
                                        time(&start2);
                                }

                                if(ToLongitude >= FromLongitude )
                                {
                                glutIdleFunc(NULL);
                                        k =1;
                                        l = 1;
                                        printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);

                                }
                                glutPostRedisplay();

                        }


                }


                /*AUV moving Vertically when FromLatitude is lesser than ToLatitude */



                if (FromLatitude < ToLatitude && j == 0 )
                {
                                FromLatitude = FromLatitude + VMR;
                                MastMotion = 0.008*FromLatitude;

                                if(difftime(end,start3) == 2)
                                {
                                        if (SteerMode == 1)
                                                printf("Time = %f Steermode LINE Lat:%f Long =
%f\n", difftime(end,start3), FromLatitude, FromLongitude);
                                        time(&start3);
                                }

                                if((FromLatitude >= ToLatitude) && (FromLongitude ==
ToLongitude))
                                {

                                        glutIdleFunc(NULL);
```

```
                                        j = 1;
                                        k = 1;
                                        printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);
                                }

                                glutPostRedisplay();

                        }


                /*AUV moving Vertically when FromLatitude is greater than ToLatitude */
                if (FromLatitude > ToLatitude && j == 0 )
                {

                        FromLatitude = FromLatitude - VMR;
                        MastMotion = 0.008*FromLatitude;

                        if(difftime(end,start4)== 2)
                        {
                                if (SteerMode == 1)
                                        printf("Time = %f Steermode LINE Lat:%f Long =
%f\n",difftime(end,start4), FromLatitude, FromLongitude);
                                time(&start4);;

                        }

                        if((FromLatitude <= ToLatitude) && (FromLongitude == ToLongitude))
                        {
                                glutIdleFunc(NULL);
                                j = 1;
                                k = 1;
                                printf("Time = %f, Steermode LINE %f, %f\n",difftime(end,start),
FromLatitude, FromLongitude);
                        }

                        glutPostRedisplay();


                }


        }
}
void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(70.0, (GLfloat) w/(GLfloat) h, 10.0, 4.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef (0.0, 0.0, -5.0);
}

void keyboard (unsigned char key, int x, int y)
```

```
{
  switch (key) {

    case 'a':
                        {
                                i = 1;
                                  glutIdleFunc(NULL);
                        }
                       break;

    case 27:
      exit(0);
      break;
    default:
      break;
  }
}
void mouse(int button, int state, int x, int y)

{
        switch(button)
                {
                        case GLUT_LEFT_BUTTON:
                                if (state == GLUT_DOWN && i == 0.0)
                                {
                                        glutIdleFunc(MoveUp);

                                }
                                break;

                }
}

FILE *WInput, *WptInput, *WAngle;
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
        WInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\WaypointInput.txt", "r");
        if( WInput == NULL )
    printf( "The file 'WaypointInput.txt' was not opened\n" );// File failed to open
        else
    printf( "The file 'WaypointInput.txt' was opened\n" );// File opened

        fseek( WInput, 0L, SEEK_SET );
        fscanf(WInput, "%f\n", &ToLatitude);//Get the destined Latitude
        fscanf(WInput, "%f\n", &ToLongitude);//Get the destned Longitude
   fclose(WInput);
        printf("ToLat = %f, ToLong = %f\n",ToLatitude, ToLongitude);

        if (ToLatitude < -1.5)
        {
                ToLatitude = -1.5;
                printf("ToLatitude value change to -1.5 as depth below that is dangerous\n");
        }
        if (ToLatitude > 1.0)
        {
```

```
                ToLatitude = 1.0;
                printf("ToLatitude value cannot be more than 1.0 the surface\n");
        }

        WptInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "r");
        fscanf(WptInput, "%f\n", &FromLatitude);//Get the current Latitude
        fscanf(WptInput, "%f\n", &FromLongitude);//Get the current Longitude
        fclose(WptInput);
        printf("Lat = %f, Long = %f\n",FromLatitude, FromLongitude);

        if (FromLatitude < -1.5)
        {
                FromLatitude = -1.5;
                printf("FromLatitude value change to -1.5 as depth below that is dangerous\n");
        }

        if (FromLatitude > 1.0)
        {
                FromLatitude = 1.0;
                printf("FromLatitude value cannot be more than 1.0 the surface\n");
        }



        WAngle = fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "r");
        fscanf(WAngle, "%f\n", &MastAngle);
        if (MastAngle > 0 && FromLatitude < 1.0)
        {
                MastAngle = 0.0;
                printf("Mast is not raised as not on surface\n");
        }
        fclose(WAngle);
        if (MastAngle > 0 && FromLatitude == 1.0)
        {
                MastAngle = 0.0;
                printf("Lower Mast before going below surface of water\n");
        }
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMouseFunc(mouse);
  glutMainLoop();
  return 0;
}
```

Rendezvous

```
/*Rendezvous module execution sequence animation*/

#include <GL/glut.h>
#include <stdlib.h>
#include<stdio.h>
```

```
#include<math.h>
#include<time.h>
#define VerticalMotionRate 0.001//Rate to move up depeding
#define HorzMotionRate 0.001 //Rate to move horizontally
#define DriftRate 0.001 //To assign a value to water current given as drift
#define StartPoint -1.0 //Keeping a record of start point
#define Steering 1//Checks whether steering mode received or not
#define Direct 1 //Steering mode direct to go straight to the point following a line
#define NotDirect 2 //Steering mode indirect
#define HOVER 1
#define CIRCLE 2
#define NONE 0
#define PI 3.14
#define Steering 1
#define LoiterTO 50
#define Loiter_TimeToLoiterPt_FCN 10
#define ToWP__Timed 1 // Timed waypoint if 1
#define TimeToWaypoint 15
#define Speed 0.35
#define ThresholdDistance 0.1
#define HMR 0.001
#define VMR 0.001
static GLfloat Helm__DistanceToPoint;
static GLfloat FromLatitude ; // The begining latitude
static GLfloat FromLongitude ; //The beginning longitude
static GLfloat MastMotion = 0.0; // Motion of the mast
static GLfloat i = 0; //Used for abort signal
static GLfloat Slope; // To get the slope of the line in direct steeringmode
static GLfloat SteerMode = Direct;//Assigning mode to steering
static GLfloat LatLongDiff; // The constant in the equation of a line for direct steermode
static GLfloat Step;
static GLfloat ToWP__LoiterType = 1;
static GLfloat Loiter = 0;
static GLfloat x, y;
static GLfloat First = 0;
static GLfloat r;
static GLfloat Theta;
static GLfloat base;
static LoiterDone = 0;
static GLfloat MTime = 0;
static GLfloat m = 0;
static GLfloat Side, Long =0;
static GLfloat TimeInState;
static GLfloat Helm =0;
static GLfloat DistanceToWaypoint;
static GLfloat ToLatitude ;
static GLfloat ToLongitude ;
static GLfloat ToWP__LoiterDuration = 5;
static GLfloat ActualTimeToWP;
static GLfloat RemainingTime;
static GLfloat DecideLoiter = 0;
static GLfloat TimeInLoiter = 0;
static GLfloat HoverPoint;
int ToWP__LoiterAtWP = 1;
int WaypointNavigator_DepthTrouble_FCN;
int Initial = 0;
```

```
int NumPoints = 2;
int j =0, k =0, n =0, l =1, p=0, q = 0;//j for lat loop, k for long loop, l for direct loop, p for indirect
int begin = 0,  count = 0;
int Hove = 0;
int iteration = 0;
int Loitering = 1;
int numIteration = 0;
float TotalTime;
double t;
time_t start, start1, start2, start3, start4, start5, start6;
time_t end, end1, end2, end3, endn, endn1;
FILE *ROutput, *ReOutput, *RTOutput;
void init(void)
{

  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_FLAT);
}

void display(void)
{
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f(0.0, 0.4, 0.8);
  glRectf(8.0, -4.0, -8.0, 2.0);
  glPushMatrix();
  glTranslatef (-1.0, 0.0, 0.0);
  glTranslatef (1.0, 0.0, 0.0);
  glTranslatef(FromLongitude, FromLatitude, 0.0);
  glTranslatef(FromLongitude, FromLatitude, 0.0);
  glPushMatrix();
  glScalef (2.0, 0.4, 1.0);
  glColor3f(0.0, 0.7, 0.4);
  glutSolidCube (1.0);
  glPopMatrix();
  glTranslatef (0.0, 0.1, 0.0);
  glTranslatef(0.08*FromLongitude, MastMotion, 0.0);
  glRotatef(0.0, 0.0, 0.0, 1.0);
  glTranslatef(0.5, MastMotion, 0.0);
  glPushMatrix();
  glScalef (0.8, 0.2, 0.5);
  glColor3f(1.0, 1.0, 0.0);
  glutSolidCube(1.0);
  glPopMatrix();
  glPopMatrix();
  glutSwapBuffers();
}


void MoveUp(void)
{
        if ((FromLongitude != ToLongitude || FromLatitude != ToLatitude) && (q == 0 || p == 0))
        {

                if(begin == 0)
                {
                        time(&start);
```

```
                        time(&start1);
                        time(&start2);
                        time(&start3);
                        time(&start4);
                        Slope =(ToLatitude-FromLatitude)/(ToLongitude-FromLongitude);
                        LatLongDiff = FromLatitude - Slope*FromLongitude;
                        begin++;
                }

                if (ToLatitude <= -2.0)
                {
                        ToLatitude = -1.5;
                }

                if(ToLatitude > 1)
                {
                        ToLatitude = 1.0;
                }

                if (FromLongitude > ToLongitude)
                {
                        FromLongitude = FromLongitude + 0.001;
                        FromLatitude = Slope*FromLongitude + LatLongDiff;
                        MastMotion = 0.008*FromLongitude;
                        q = 1;
                        if(difftime(end,start1) == 2)
                        {
                                printf("Time = %f, Steermode LINE %f, %f\n",difftime(end,start),
FromLatitude, FromLongitude);
                                time(&start1);
                        }
                        if(ToLongitude >= FromLongitude)
                        {
                        //glutIdleFunc(NULL);
                                l = 1;
                                p = 1;
                                printf("Hello Time = %f Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);
                        }
                        glutPostRedisplay();
                }

                if(FromLongitude < ToLongitude)
                {
                        FromLongitude = FromLongitude + 0.001;
                        FromLatitude = Slope*FromLongitude + LatLongDiff;
                        MastMotion = 0.008*FromLongitude;
                        q = 1;
                        if(difftime(end,start2) == 2)
                        {
                                printf("Time = %f, Steermode LINE %f, %f\n",difftime(end,start),
FromLatitude, FromLongitude);
                                time(&start2);
                        }
                        if(ToLongitude <= FromLongitude)
                        {
```

```
		//		glutIdleFunc(NULL);
					p = 1;
					l = 1;
					printf("Time = %f Steermode LINE %f, %f\n",difftime(end,start),
FromLatitude, FromLongitude);
				}
						glutPostRedisplay();
			}
			if((FromLongitude == ToLongitude))
			{
				if(FromLatitude < ToLatitude && q == 0)
				{
					FromLatitude = FromLatitude + VMR;
					MastMotion = 0.008*FromLatitude;
					if(difftime(end,start3) == 2)
					{
						printf("Time = %f Steermode LINE Lat:%f Long = %f\n",
difftime(end,start), FromLatitude, FromLongitude);
						time(&start3);
					}
					if(FromLatitude >= ToLatitude)

					{
						p = 1;
						q = 1;
						printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);
					}
					glutPostRedisplay();
				}

				if(FromLatitude > ToLatitude && q == 0)
				{
					FromLatitude = FromLatitude + VMR;
					MastMotion = 0.008*FromLatitude;
					if(difftime(endn,start4) == 2)
					{
						printf("Time = %f Steermode LINE Lat:%f Long = %f\n",
difftime(end,start), FromLatitude, FromLongitude);
						time(&start4);
					}
					if(FromLatitude <= ToLatitude)

					{
						p = 1;
						q = 1;
						printf("Time = %f, Steermode LINE %f,
%f\n",difftime(end,start), FromLatitude, FromLongitude);
					}
					glutPostRedisplay();
				}
			}
		}

			if (FromLongitude == 0 && ToLatitude == 0 && ToLongitude == 0 && FromLatitude
== 0)
```

```
		{
			begin = 1;

		}

		if (Loitering == 1 && ((FromLongitude == 0 && ToLatitude == 0 && ToLongitude ==
0 && FromLatitude == 0) || (q == 1 && p == 1)) && LoiterDone == 0)
		{
			//printf("Hello\n");
			if(begin == 1)
			{
				printf("Time to Loiter = %f, Lat = %f, Long = %f\n", difftime(end,
start), FromLatitude, FromLongitude);
				time(&start5);
				time(&start6);
				begin++;
			}

			if ((NumPoints<2 && LoiterTO == 0 || ToWP__LoiterType == NONE) &&
LoiterDone == 0)
			{
				LoiterDone = 1;

				printf("No time to Loiter \n");
			}

			if ((NumPoints<2 && LoiterTO>0) || ToWP__LoiterType == HOVER &&
LoiterDone == 0)
			{

				if (count == 0)
				{
					HoverPoint = FromLongitude;
					printf("HOVERING, HoverPoint = %f, Hove = %d \n",
HoverPoint, Hove);
					count++;
				}

		if (FromLongitude < (HoverPoint + 1.0) && Hove == 0 && LoiterDone == 0)
			{

				FromLongitude = FromLongitude + 0.001;


				if (FromLongitude >= HoverPoint + 1.0)
				{
					++Hove;

					printf("Hove in fwd= %d, Time Hovering = %f\n", Hove,
difftime(end1, start5));
				}
				if ((difftime(end1, start5) >= (LoiterTO-Loiter_TimeToLoiterPt_FCN-
20)))
				{
					LoiterDone = 1;
```

```
                                        printf("LoiterDone\n");
                                }
                        }
                        if (FromLongitude > HoverPoint  && Hove == 1 && LoiterDone == 0)
                        {
                                FromLongitude = FromLongitude - 0.001;
                                if (FromLongitude <= HoverPoint)
                                {
                                        --Hove;

                                        printf("Hove in bwd= %d Time Hovering = %f \n",Hove,
difftime(end1, start5));
                                }
                                if ((difftime(end1, start5) >= (LoiterTO-Loiter_TimeToLoiterPt_FCN-
20)))
                                {
                                        LoiterDone = 1;

                                }
                        }

                        glutPostRedisplay();
                }
                time(&end1);

                if (NumPoints>=2 && LoiterTO>0 && ToWP__LoiterType == CIRCLE &&
(difftime(end2, start5)<=LoiterTO-Loiter_TimeToLoiterPt_FCN) && Helm ==0)
                {
                        Step = 360/NumPoints;
                        for (m=0; m<Step; m=m+1)
                        {
                                if (First == 0)
                                {
                                        First = 1;
                                        r = pow(pow((FromLatitude-ToLatitude),2) +
pow((FromLongitude-ToLongitude),2), 0.5);
                                        Helm__DistanceToPoint = r;
                                        if (Helm__DistanceToPoint <= 0.25)
                                        {
                                                glutIdleFunc(NULL);
                                                printf("Distance to point is less than 20 m\n");
                                                Helm++;
                                        }
                                        Side = pow(pow((FromLatitude-ToLatitude-r),2) +
pow((FromLongitude-ToLongitude),2), 0.5);
                                        Theta = 2*asin(Side/(2*r));
                                        printf("r=%f, Theta = %f\n",r, Theta);
                                        FromLongitude = -r * cos(Theta) + ToLongitude;
                                        FromLatitude = -r * sin(Theta) + ToLatitude;
                                        glutPostRedisplay();
                                        m++;
                                        printf("FromLat = %f, FromLong = %f, m = %f\n",
FromLatitude, FromLongitude, m);
                                }

                                if (m>0 && LoiterDone == 0)
```

```
                                        {
                                                Theta = Theta + 0.00001;
                                                FromLongitude = -r * cos(Theta)+ ToLongitude;
                                                FromLatitude = -r * sin(Theta)+ ToLatitude;

                                                if (difftime(end2, start1) == 2)
                                                {
                                                        printf("TimeInState = %f, FromLat = %f, FromLong
= %f\n", difftime(end2, start), FromLatitude, FromLongitude);
                                                        time(&start1);
                                                }

                                                if (difftime(end2, start)>=LoiterTO-
                                                Loiter_TimeToLoiterPt_FCN)
                                                {
                                                        LoiterDone = 1;
                                                }

                                                if(FromLatitude > 1.0)
                                                {
                                                        FromLatitude = 1;
                                                }
                                                if(FromLatitude < -1.5)
                                                {
                                                        FromLatitude = -1.5;
                                                }

                                                glutPostRedisplay();
                                        }


                                }
                                time(&end2);

                        }
                }
                time(&end);
                if((Loitering == 0 && p == 1 && q == 1) || (LoiterDone == 1) )
                {
                        if(numIteration == 0)
                        {
                                ROutput = fopen( "C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "w"
);
                                fprintf( ROutput, "%f\n", FromLatitude );
                                fprintf( ROutput, "%f\n", FromLongitude );
                                fclose(ROutput);

ReOutput = fopen( "C:\\Research\\Animation\\Temp\\Mission\\RendezvousOutput.txt", "w" );
                                fprintf( ReOutput, "%f\n", FromLatitude );
                                fprintf( ReOutput, "%f\n", FromLongitude );
                                fprintf(ReOutput, "%f\n", difftime(end, start));
                                fclose(ReOutput);
                RTOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "r");
                                fscanf(RTOutput,"%f\n", &TotalTime);
                                fclose(RTOutput);
```

```
                        TotalTime = TotalTime + difftime(end, start);

        RTOutput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "w");
                        fprintf(RTOutput, "%f\n", TotalTime);
                        fclose(RTOutput);
                        numIteration++;
                        exit(0);
                }
        }
}


void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(70.0, (GLfloat) w/(GLfloat) h, 10.0, 4.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef (0.0, 0.0, -5.0);
}

void keyboard (unsigned char key, int x, int y)
{
  switch (key) {

    case 'a':
                 {
                        i = 1;
                          glutIdleFunc(NULL);
                 }
                break;

    case 27:
      exit(0);
      break;
    default:
      break;
  }
}
void mouse(int button, int state, int x, int y)

{
        switch(button)
                {
                        case GLUT_LEFT_BUTTON:
                                if (state == GLUT_DOWN && i == 0.0)
                                {
                                        glutIdleFunc(MoveUp);

                                }
                                break;
                        }
}
```

```
FILE *RInput, *ReInput, *RTInput;
int main(int argc, char** argv)
{
  glutInit(&argc, argv);

        RInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Position.txt", "r");
        if( RInput == NULL )
    printf( "The file 'Position.txt' was not opened\n" );// File failed to open
        else
    printf( "The file 'Position.txt' was opened\n" );// File opened

        fseek( RInput, 0L, SEEK_SET );
        fscanf(RInput, "%f\n", &FromLatitude);//Get the destined Latitude
        fscanf(RInput, "%f\n", &FromLongitude);//Get the destned Longitude
   fclose(RInput);

        ReInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\RendezvousInput.txt", "r");
        if( ReInput == NULL )
    printf( "The file 'RendezvousInput.txt' was not opened\n" );// File failed to open
        else
    printf( "The file 'RendezvousInput.txt' was opened\n" );// File opened

        fseek( ReInput, 0L, SEEK_SET );
        fscanf(ReInput, "%f\n", &ToLatitude);//Get the destined Latitude
        fscanf(ReInput, "%f\n", &ToLongitude);//Get the destned Longitude
   fclose(ReInput);
        RTInput = fopen("C:\\Research\\Animation\\Temp\\Mission\\Time.txt", "r");
        fscanf(RTInput,"%f\n", &TotalTime);
        fclose(RTInput);
        printf("Mission Time = %f\n", TotalTime);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMouseFunc(mouse);
  glutMainLoop();
  return 0;
}
```

Vita

- Date and place of birth: 08/ 14/ 1978, City: Calcutta, State: West Bengal, Country: India

- Educational institutions attended and degrees already awarded

  Master of Science, Electrical Engineering Iowa State University, Ames, Iowa, May2003

  Bachelor Of Engineering, Electrical and Electronics Engineering, Birla Institue Of Technology, India, May2001

- Professional positions held

  Assistant Professor Kentucky State University Fall2005 – Present

  Visiting Assistant Professor Kentucky State University Spring 2005 (Jan 18 – May15)

  Applied Research Lab, Penn State University, State College, Pennsylvania, Summer 2004, Research assistant

  Discrete Event System Lab, University of Kentucky, Lexington, Kentucky, Fall 2001 –Spring 2002, Summer 2003 – Spring 2004, Fall 2004, Research assistant

  Iowa State University, Ames, Iowa, Fall 2002 –Spring 2003
  Teaching assistant

- Scholastic and professional honors

  Awarded best presenter in session of Discrete Event System at American Control Conference (ACC), July 2004, Boston, Massachusetts

  Awarded Student travel grant of $600 to attend American Control Conference (ACC), July2004, Boston, Massachusetts

  Awarded Kentucky graduate scholarship by University of Kentucky

  Funded by University Of Kentucky Electrical Engineering Department for Ph.D as research assistant.

Funded by Iowa State University Electrical Engineering Department for Masters as teaching assistant

Awarded scholarship under National Scholarship Scheme for securing 100% marks in mathematics in the All India Secondary School Examination 1995

Awarded certificate for securing 91% in the National Mathematics Olympiad (India), 1995.

Awarded certificate for securing above 80% marks in The Indian Association Of Physics Teachers National Standard Examination in 1997

- Professional publications

"Adaptive Supervisory Control of Discrete Event Systems", V. Chandra and S. Bhattacharyya, Instrumentation, Systems and Automation, Chicago, Oct 2005.

"On the Rapid Reconfiguration of Modular Manufacturing Systems", V. Chandra, S. Bhattacharyya **a**nd S. Mohanty, Intelligent Processing and Manufacturing of Materials, Montererey (IPMM) July 2005.

"A Systematic Methodology for Actuator Augmentation in the Supervisory Control of Discrete Event Systems",V.Chandra and S. Bhattacharyya, 11th International Conference on Information Systems Analysis and Synthesis: ISAS´05 and 2nd International Conference on Cybernetics and Information Technologies, Systems and Applications: CITSA'05, Orlando, USA , July 14-17, 2005

"The Design of Adaptive Supervisors for Discrete Event Systems", has been accepted for presentation at the 9th World Multiconference on Systemics, Cybernetics and Informatics (WMSCI 2005) Orlando, USA, July 10-13, 2005

"Hybrid-Model based Hierarchical Mission Control Architecture for Autonomous Undersea Vehicles", S. Tangirala, R. Kumar, S. Bhattacharyya, M. O'Connor, and L. E. Holloway, American Control Conference (ACC), June 2005

"A Discrete Event Approach to Network Fault Management", S.Bhattacharyya, Z.Huang, V.Chandra and R.Kumar. American Control Conference (ACC), Boston, July 2004.

"Diagnosis of Discrete Event Systems in Rule-Based Model Using First order Linear Temporal Logic" Z.Huang, S.Bhattacharyya, V.Chandra and R.Kumar. American Control Conference (ACC), Boston, July 2004.

"Automatic Extraction of Discrete Event System Controllers". V. Chandra and S. Bhattacharyya. The 8th World Multiconference on Systemics, Cybernetics and Informatics (SCI), Orlando, FL, July 2004.