DESIGN AND DEVELOPMENT OF COMMAND AND CONTROL SYSTEM FOR AUTONOMOUS UNDERWATER VEHICLES

By
TAN YEW TECK

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING
AT
NATIONAL UNIVERSITY OF SINGAPORE
SINGAPORE
OCTOBER 2008

Table of Contents

List of Tables

List of Figures

Abstract

Over the last decades, the problem of building Autonomous Underwater Vehicles (AUVs) for missions in partially unknown underwater environment continue to challenge researchers. Although the AUV technology has matured and commercial systems have appeared in the market, a generic yet robust AUV command and control system still remains a key research focus. This thesis presents a novel command and control system architecture for modular AUVs. Particular focus on this thesis is the design and development of a generic control and software architecture for a single modular AUV while allowing natural extensions to multi-vehicle scenarios.

The proposed command and control (C2) system has a hybrid, modular hierarchical control architecture. It adopts deliberative top-down approach in mission level decision making and task planning while utilizing reactive bottom-up approach for navigational control, obstacle avoidance and vehicle fault detection. The structure provides vehicle developers with an explicit view of the clearly defined control responsibilities at different level of control hierarchy. The underlying software architecture of the C2 system adopts component and modular based design principle. Every C2 component has its local data structure and implements its own logic without interfering with other components. Such a design has several advantages for component construction and maintainability. All the components have a uniform software interface to facilitate inter-component communication within the AUV via Remote Procedural Call (RPC). This allows computational load distribution by deploying C2 components onto different processors in the AUV. The component based approach exhibits robustness and adaptability as different components can be configured or exchanged depending on the requirement of mission.

The resultant C2 system has been programmed and tested on a 3D System-In-The-Loop simulator. It is also operational on a prototype AUV known as STARFISH built by the Acoustic Research Laboratory (ARL) of the National University of Singapore

(NUS). The resultant C2 system is utilized in a simple navigational mission in the simulator and for a surface run from a lake test using the prototype AUV.

Acknowledgments

This thesis could not have been completed without the help of several important people. First, I would like to thank my supervisor, A/Prof. (Dr.) Prahlad Vadakkepat and my co-supervisor, Dr. Mandar Chitre for their technical guidance and for sacrificing so much of their personal time to help me in completing this work. I would also like to express my gratitude to the members of the Command and Control group for the STARFISH project for being so helpful through the project period.

Not forgetting the members of Control and Simulation Laboratory and Acoustic Research Laboratory (ARL), thank you for sharing their technical knowledge in one way or another for Autonomous Underwater Vehicle (AUV) missions. In particular, I greatly appreciate the help of Mehul Sangekar and Shiraz Shahabudeen for not only helping me in working with the AUV but also for giving me valuable feedback on my work.

Finally, I would also like to thank my foster father, Brian Kelly and my family for their love and support throughout this past year.

Chapter 1

Introduction

This thesis presents a novel command and control (C2) system developed for modular Autonomous Underwater vehicle (AUV) performing autonomous underwater surveying missions. C2 system of an AUV is the key component that determines the outcome of an autonomous mission. While AUV technology has matured over the last few years, AUV's C2 system still remains a challenge for researchers.

The research motivations are discussed in Section 1.1 and some applications of the AUV are illustrated in Section 1.2. Section 1.3 presents the problem statement and adopted approach and Section 1.4 provides the outline of this thesis.

1.1 Motivation

Despite substantial progress in Autonomous Underwater Vehicles (AUVs) technologies over the last few years, the Command and Control (C2) system continues to challenge researchers. To carry out a mission, the C2 system must be robust, adaptive, and able to cope with the changes in dynamics and uncertain environments. The C2 system is a highly complex and critical software in a mission-based AUV. At a higher level, it is in charge of defining mission tasks based on a predefined mission

file, interpreting mission commands from the operator, making decisions and taking appropriate actions if a problem is encountered to ensure the safety of the AUV throughout the mission execution. At a lower-level, the C2 system must be capable of interpreting raw data coming from the AUV's sensors and combining different actuators to generate the desired behavior to fulfil each mission task.

The C2 system for AUV projects has been evolving throughout the years. The early development in C2 systems had architectures adopting reactive or deliberative, centralized or distributed, top-down or bottom-up approaches. As AUV technologies advanced, the need for better functionalities and capabilities arised in the AUV's working environment. Majority of the C2 systems nowadays utilize hybrid architectures. Hybrid architectures are constructed by the combination and/or integration of two or more different architectures that takes the advantages of each of the architectures while minimizing their individual weaknesses.

The current trend in mobile robotics software is to move towards component-oriented design principle. It has proven to be an effective approach in developing robotic software [7]. There are already a few frameworks available for mobile robots as well as the industrial applications [28, 3]. However, few are specifically designed for C2 purposes. Although the C2 systems are usually hierarchical in nature and different modules within the overall control architecture have very distinctive tasks and responsibilities, implementing the underlying software architecture based on component-oriented principle provides certain advantages. Since every module is different, specialized and well-defined, the software component interface can be designed for each individual module in the C2 system. This allows software developers to implement different logics and algorithms in the same component without

affecting the overall architecture. The pre-defined interface of components also encourages modularity of system tasks and responsibilities and makes construction and maintenance of C2 system easier and manageable.

Instead of developing complex, expensive monolithic AUVs for underwater missions, researchers nowadays are moving their attention towards building simpler, low-cost modular AUVs. Modularity in AUV's development at software and hardware level provides benefits to the developers and users. Different sections of AUVs can be built separately by different group of developers at the same time provided they comply to the same hardware interfaces. Besides that, different AUV sections can also be exchanged or added to provide the functionalities needed for a particular mission task at the mission site. Every changeable section has its own software modules that implements different algorithms depending on the section's responsibilities in the overall AUV setup, and when put together, they form a complete working AUV. However, this plug-and-play capability can only be achieved if the underlying C2 system is capable of adapting to the various AUV configurations for different missions.

1.2 The STARFISH Project

The target application of the research described in this thesis is the C2 system for prototype AUVs used in Small Team of Autonomous Robotic Fish (STARFISH) project. The STARFISH project is an initiative at the Acoustic Research Laboratory (ARL) of the National University of Singapore (NUS) to study collaborative missions carried out by a team of low-cost, modular AUVs.

The use of a team of AUVs has many advantages compared with a single complex AUV. A team of AUVs provides redundancy as well as fault tolerance; failure in one of

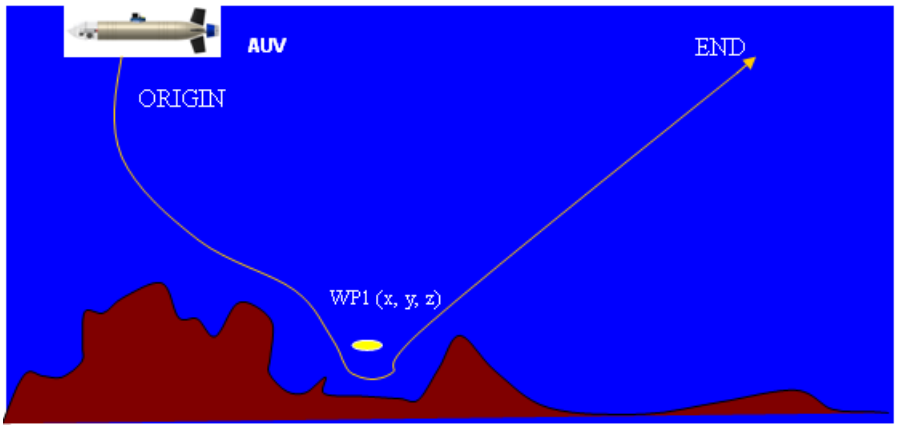


Figure 1.1: Scenario for mission 1. The AUV goes from ORIGIN point at the surface to END point via waypoint (x,y,z) within 2 km from the surface.

the AUVs will be less likely to affect the outcome of a mission compared with a single AUV. Besides that, a team of distributed AUVs is able to provide larger mission area coverage as well as simultaneous spatial sampling, thus, results in shorter mission time and lower mission cost.

In order to demonstrate the functionalities of the project's final outcome, two missions have been defined to validate the resulting prototype AUVs as well as confirming their underlying C2 system's capabilities in carrying out autonomous missions in single and multi-AUVs scenarios:

Mission 1 - Single AUV Navigation Capability: In this mission, all the basic functionalities of a single AUV is tested. This includes the hardware and software components in the AUV's modules. Given a chart, an AUV is instructed to go from location ORIGIN (on surface) to location END (destination on surface) via waypoint WP1 (x, y, z) within 2 km from the surface in a specified period of time (Fig. 1.1). Between location ORIGIN and location END, the water depth is more than 5m. The AUV should avoid obstacles shown in the chart.

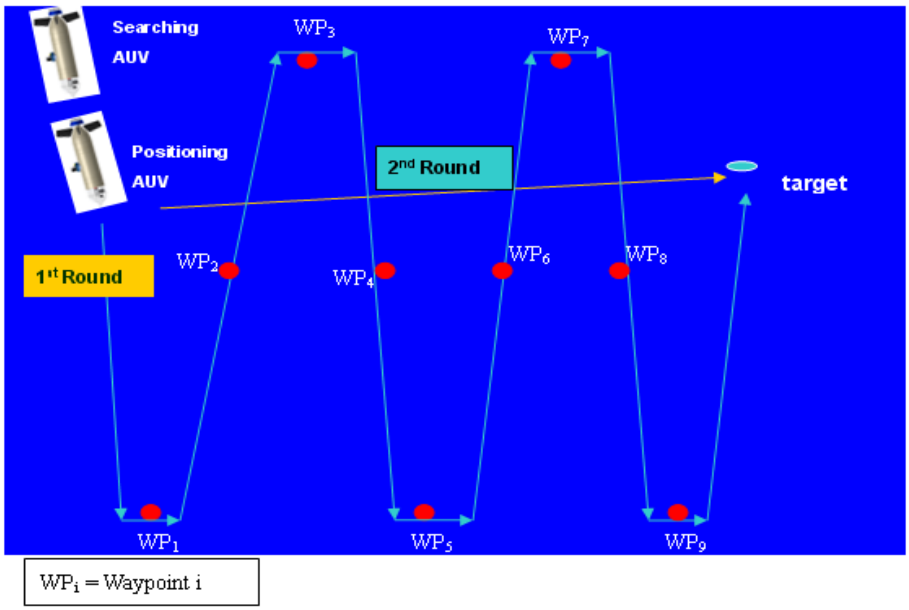


Figure 1.2: Scenario for mission 2. Two AUVs each with different payload section work together to locate the target. Once the target is found, one of the AUVs will go back to re-acquire the target based on the recorded position.

Mission 2 - Multi AUV Target Re-acquisition: This mission is designed to test the collaborative behavior in a multi AUV scenario. A team of 2 AUVs, one specialized in survey scanning, the other specialized in position acquisition, are deployed and assigned to search a 500m square area containing a reflective target ($\tilde{1}$m) on the sea bed (Fig. 1.2). AUV specialized in position acquisition navigates within the communication range of the AUV specialized in survey scanning to provide better estimation of positioning. Once the target is located, the AUVs move away from the target and surface at a pre-determined location. One of the AUVs then go back to re-acquire the target based on the recorded position.

Mission Requirements: The two missions described above defined the mission requirements for the onboard C2 system. First, the mission involves mission and path planning from the ORIGIN point to the END point before the navigational command

can be generated. Second, in order to ensure the AUV's safety throughout the mission, any obstacle lying in the AUV's path must be detected and avoided. However, in case the AUV is unable to avoid an obstacle, decision has to be made to abort the mission. Third, in missions where multi-AUVs are involved, there must be communication among the AUVs to exchange information. Furthermore, all the AUVs must be able to keep track of their mission progress and update the operator whenever they can. Finally, to make sure the team of AUVs carry out a mission effectively, a mission must be broken down into individual tasks where each can be handled by the AUV with its specialized payload section. This can be achieved through the interaction among the AUVs in the team and task assignment can be done according to the multiple AUV's setup.

1.3 Problem Statement

The research in this thesis concentrates on developing generic C2 system for a single modular AUV in STARFISH project while allowing natural extension to be used in multiple AUVs working as a team. The C2 system's construction is divided into two parts: the control architecture and software architecture.

Control architecture refers to the organization of mission tasks into individual control entities at different levels of control hierarchy. This includes mission translation into individual primitive navigational tasks; maneuver commands generation; mission progress and vehicle safety monitoring; and mission level as well as vehicle level decision making. Based on the mission requirements mentioned in section 1.1, the proposed C2 system's control architecture must be able to handle all the mission tasks with minimum intervention from the operator.

On the other hand, software architecture refers to the overall C2 system structure which comprises of the software components, internal and external properties of those components and the relationship among them. In this thesis, a modular software architecture is desired to allow individual control entities to be implemented as software components. Each software component handles specified command and control tasks while interacting with other components within the architecture to achieve the mission requirements. The C2 system's software architecture is built based and on top of the work descried in [9].

1.4 The Thesis Layout

This thesis is organized as follows. Chapter 2 provides a brief discussion of related works in design, and the development of control and software architectures for mobile robot and AUVs. Chapter 3 presents the novel control architecture developed for the AUV's C2 system. Chapter 4 illustrates the software architecture of the proposed component based C2 system. Chapters 5 and 6 present the simulation results from an in-house built System-In-The-Loop simulator over a number of simulation cases and a brief description of the first AUV prototype as well as its lake test experiment. Finally, Chapter 7 concludes the thesis and makes suggestions for future work.

Chapter 2

Background

Developing the C2 system or mission controller for autonomous and remotely operated robotic systems is a challenging task for researchers. It has to be robust and flexible in handling uncertainties and animosities that might arise during the robot's operation in a highly hazardous and unknown environment. For the past few years, a great number of autonomous mission controllers has been developed and implemented in autonomous underwater, ground or air robotic systems. A complete C2 system is described by its control architecture and the underlying software architecture.

2.1 Command and Control Architecture

Command and Control system architecture generally adopts the following architectures: reactive or deliberative reasoning; distributed or centralized processing; command arbitration or sensor fusion; top-down or bottom-up control [27]. However, due to the requirement of self-supervisory, goal-oriented and complex nature of autonomous mission, most of the mission controllers adopt a hybrid approach which integrates different architectures to utilize the advantages of some of the architecture while minimizing the limitations of others [4].

In [27] Yavuz adopted a hybrid approach that utilizes reactive, deliberative, distributed and centralized control for autonomous mobile robots. The author applied fuzzy logic for centralized command arbitration by integrating activated behaviors from distributed decision making processes running asynchronously across the robotic system. The control architecture consists of deliberative modules that make high-level navigational and tasks planning, and low-level reactive modules generating reflexive and responsive reactions based on the inputs from sensors and actuators. The division of the control tasks into individual modules that are distributed across different level of control hierarchy increased the robustness, flexibility and adaptability of the resulted control system.

For AUV mission controllers, Ridao et al [23] reports the implementation of three layers of hierarchical mission controller which combined deliberative and reactive control architecture in their semi-AUV, SAUVIM, to allow both predictability and reactivity. Its hierarchical structure is produced by the planner according to the world model in order to get a predictable scheme of the execution of the mission while the parallel execution of the task modules coordinating sensors and actuators guarantees reactivity. In 2007, Ridao et all [17] continued their development in AUV technology and came out with another control architecture called O^2CA^2. There are three levels in this control architecture where the "Vehicle level" implements the vehicle controller while the "Task level" coordinates the mission tasks. This approach decouples the robot control from the robot guidance, and together they exhibited reactive behaviors. On the other hand, the "Mission level" behaves deliberatively where it defines the high level mission tasks as well as the configuration for the task level to accomplish each mission task.

Elsewhere, Bhattacharyya et al [5] implemented a hybrid mission controller for AUV simulation for rapid development, while Yavnai [26] developed a reconfigurable mission controller called ARICS that combines the characteristic of both reasoning-based and reactive-reflexive behavior to provide goal-directed planning and good responsiveness.

From the literature, it can be observed that majority of the command and control system developed for mobile ground or underwater robots is hybrid in its nature. Such an approach which incorporates both deliberative and reactive behavior has demonstrated the robustness, flexibility, adaptability and extendability that is required for building complex robotic systems which are capable of handling various situations and uncertainty in a highly dynamic environment.

Deliberative architecture [18] is both hierarchical and top-down in its control structure. Planing and decision making are done at the upper level and passed down to the lower level for execution. Deliberative architecture relies heavily on the information of the world model where the vehicle is situated in. During a mission, raw data from the sensors are processed and used to update the model. The dynamically acquired and updated model is then used for new plans or actions when necessary. To handle problems in dynamic and partial unknown environment with the latest acquired information is desired for AUV navigation. However, such an approach suffers from computational latency during the sense-model-plan-act process.

Reactive architecture [4] is also known as bottom-up or behavioral architecture. It consists of a set of elemental behaviors that defines the AUV's capabilities. Global behaviors emerge from the combination of several elemental behaviors activated in

parallel when interacting with the world. Behavioral architecture reacts to the environment directly without involving any high level reasoning or replanning process. Data is taken directly from the sensors to evaluate the current world model and appropriate behaviors are chosen to adept to the model. The sense-react principle is suitable for operations in highly dynamic world. However, this architecture may lead the AUV into local minima because only immediate sensing is utilized to react with the environment.

2.2 Component Based Software Architecture

Software architecture defines the organization as well as the construction of software modules for a system. Component-based Software Engineering is becoming the popular approach adopted by robotists in developing robot software around the world in the last few years. Such a software engineering approach enforces functional modularization which helps control dependencies, distributed implementation and increase system flexibility and robustness. Among the frameworks that are available including Orca [6] and Orocos [8].

Orca is an open-source Component-Based software engineering framework designed for mobile robots. It comes with an online repository that provides free software components for building mobile robots. The framework emphasizes on and provides two main advantages: software modularity and re-usability. By adopting principle of modularity in software design and development, the resulted system is easily reconfigurable to satisfy the system requirements due to the explicit and controlled software dependencies. Besides that, it also allow the development work to be distributed among individual components involved since one component is not

directly interacting with another component. Furthermore, its software re-usability is achieved through re-using the existing modules across a project that are either sourced externally or built in house, in which the development cost and time can be greatly reduced.

Orocos which stands for Open Robot Control Software, aims to develop a general-purpose, free software, and modular framework for robot and machine control. This framework consist of three major types of modules/components: Supporting modules which is software without functional robotics contents, but provides visualization and simulation for building the software.The Robotics modules which implement specific robotic algorithms and User modules which build and configure the complete robotic system based on the two modules mentioned before. This framework provides the flexibility for the developer to build robotic systems based on the type of modules associated with. Under the open source license, it also enables developers to implement their own stand-alone components which can be adapted and extended by other end-users.

In AUV research, developers have started to adopt modular based software developments for the control system. Early efforts spotted in this area are reported in [24]. In the paper, Rodseth applied Object-Oriented software development principle in building the control system for AUV. The components of the control system are identified as objects. Each object is a combination of its private state and methods to manipulate it. The division of system tasks into individual objects encourage modularity of the resulted system. It also promotes re-usability and reliability through generalization of object definitions in class trees which helped to factor out common operations and decreases the number of coding errors.

Recent developments in AUV control system have adopted component based design principle as in Neptus [10] and MOOS [20]. Neptus is a mission planning and specification framework for AUV. Its is composed of individual service-oriented software components that provide mission specified services across a network. Each component keeps track of their own internal state and behaves according to the component's current state. Besides that, a component in Neptus can consist of several sub-components which work together to handle complex tasks like mission planning or mission reviewing and analysis.

MOOS on the other hand, is a set of key processes running in distributed components to fulfill the ubiquitous roles in mobile robotics. It has been successfully used to build the command and control system of AUV for research missions. The components are designed to handle different mission tasks, and are connected in a star-like topology through a central database/server. Each component implements its own algorithm and has no knowledge of the content of other components. There is no peer to peer communication among the components and, the sharing of data and information is facilitated by the central database. Although this design is vulnerable to communication bottle-necking at the centralized server, it keeps the network simple regardless of the number of components that are connected to it. Besides that, the server has complete knowledge of all active components which not only makes communication resources allocation feasible, but also prevent badly designed components from directly interfering with other component.

Different from the works mentioned above, the proposed architecture clearly defines navigational, mission and vehicle fault detection tasks into individual component each with its own mutual assigned responsibilities. The components are modeled as

agents which work together to fulfill the command and control tasks that are required in an AUV mission. Besides that, the proposed control and software architecture also allows changeable components for specified mission tasks and various AUV setups. The aim is to develop a command and control system that can be deployed in modular AUVs and in the future, allows natural extension and simple modification for multi-AUV scenarios.

Chapter 3

Command and Control System Architecture

3.1 Introduction

This chapter presents a novel command and control system architecture for the AUVs. An overview of the architectural design is illustrated in Fig 3.1. This is followed by a detailed description of all the components distributed among the three levels of control hierarchy: Supervisory level, Mission level and Vehicle level. There are total of seven components in the control architecture that interact with each other to carry out assigned missions: The Captain, Safety_Officer and Cheif_Scientist component are categorized under the Supervisory level (Section 3.2); the Task_Planner component is located at the Mission level (Section 3.4) and finally the Path_Executor, Obstacle_Detector, Chart_Checker and Scientist components are at the Vehicle level.(Section 3.5).

3.2 The C2 Architecture

Command and control system perform tasks ranging from planning, coordinating, directing and controlling varies activities in an AUV. It receives the processed data from the sensors as inputs and then outputs the control commands to the actuators to generate desired maneuver behavior to achieve the mission objective while keeping the AUV safe throughout the mission execution. The review of the literature revealed various control architectures implemented by different researchers in the field in which hybrid architecture is the most popular. Hybrid architecture is constructed by the combination of both deliberative and reactive architectures.

In STARFISH project, a novel C2 system based on hybrid hierarchical control architecture has been developed (Fig. 3.1). This means it adopts a deliberative-reactive architecture while having the control modules arranged in hierarchical order to depict the different levels of command responsibilities. As proposed by researchers [27, 17, 13], our architecture consist of three levels: Supervisory level, Mission level and Vehicle level. The Supervisory level is in charge of commanding and monitoring the high level mission and vehicle status while ensuring the vehicle's safety throughout the mission. The Mission level is responsible for mission planning and finally, the Vehicle level carries out the mission tasks and performs obstacle avoidance by utilizing different Sentuators (sensors and actuators) to generate the desired maneuvering behaviors. An external communication component (Signaling_Officer) has been built to provide communication link with the mothership/operator or with another AUV. Chart Room is the database where all the maps of mission areas are stored while the Mission Script consists of different mission files identified by their mission numbers.

The following sections provide detailed descriptions regarding the responsibilities

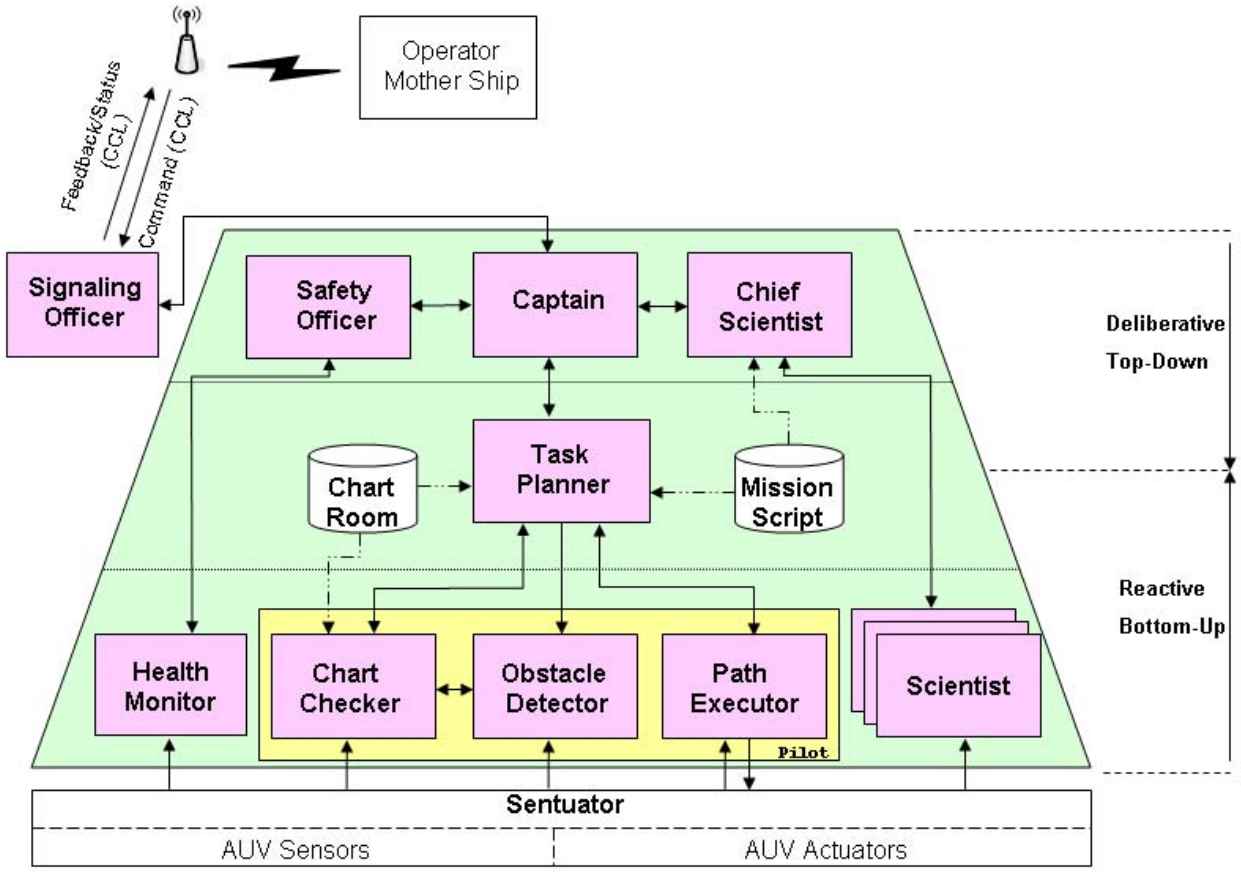


Figure 3.1: Hybrid Control Architecture for the AUV.

and tasks of different components in different levels of control:

3.3 Supervisory Level

There are three components under the Supervisory level: the Captain, the Chief_Scientist and the Safety_Officer. Components in this level carry out the main decision making issues regarding the navigation, mission and vehicle safety. Any one of these components has the right to modify or abort a mission when necessary.

3.3.1 Captain

The Captain component is in charge of the high level supervisory tasks. It starts, coordinates, oversees and controls the execution of all other components while keeping track of the mission progress. Every component in C2 keeps a record of their internal state. When a start command is received from the operator, the Captain checks the internal state to make sure that all the components are in STANDBY state before passing the command down to the Task_Planner. In situations where the AUV encounters problems caused by software errors or hardware failures, the Captain determines the source of the problem and attempt to solve it by either resetting the component's internal states or restarting the component if the first attempt fails. However, if the problem continues to exist, the current mission is aborted and the operator is notified. The decision making is performed based on inputs from the components in the C2 system with a simple rule-based system with knowledge represented as IF-THEN type predefined rules. For missions that involve multi-AUVs, the Captain can also be involved in realizing cooperation and coordination among the AUVs.

3.3.2 Chief_Scientist

The STARFISH AUV can have different payload sections added or exchanged to meet the requirements of the mission and to perform underwater scientific experiments. In the proposed C2 system, the Chief_Scientist is responsible for the command and control of payload sections. It detects payload sections attached to the AUV and based on the mission tasks specified in the mission file, controls and coordinates the Scientist components at the vehicle level. During mission planning, the Cheif_Scientist checks the AUV configuration file for any attached payload section and tries to establish communication with the payload sections. It then retrieves the task sequence denoted by the *payload* sub-tag in the mission file (Table 3.1). When the AUV is in the mission area, the Chief_Scientist turns on the corresponding Scientist components and starts analyzing the obtained information. When necessary, the Chief_Scientist may inform the Captain to modify its navigational plan to new mission area, or to abort the current mission if it fails to perform the assigned tasks.

3.3.3 Safety_Officer

Due to the high cost involved in developing an AUV, it is important to ensure the AUV's safety throughout a mission execution. For an autonomous mission, the AUV must be able to detect any abnormality that might arise and take necessary steps to make sure that it is not lost during the mission. The safety issues in our C2 system are divided into two categories: vehicle safety and mission safety. Both the safety issues are handled and monitored by the Safety_Officer component. This component resides at the supervisory level of the C2 control hierarchy because whenever an emergency occurs, it can take over the control of the vehicle without going through the Captain

component.

Vehicle safety refers to the health status of the vehicle's hardware. This includes all the sensors and actuators in the vehicle. Every hardware device in the vehicle maintain its own health status, and it is constantly queried by the Health_Monitor Component. For every component's time tick, the Safety_Officer pulls data regarding the health conditions of all the devices from the Health_Monitor component. It then analyzes the health condition of each device and informs the Captain if any malfunction are detected. However, whenever a critical situation like water leak happens, the Safety_Officer will cut off the entire AUV's power and drop the ballast to prevent the hardware from being damaged.

Mission safety concerns with the safeness of mission generated commands, the actions performed by the vehicle and the vehicle's location throughout the mission execution. Since the Safety_Officer checks the data returned from sensors and actuators periodically, any dangerous behavior or action resulted from mission related issues will be detected. For example, the command resulted from path planning with the presence of obstacles might cause the vehicle to navigate at a depth that is over the vehicle's safety limit. Once the vehicle is deeper than the limit, data returned from depth sensor will set off the alarm in the Safety_Officer component and the mission will be aborted. Under the influence of sea current and underwater turbulence , the vehicle might maneuver with large pitch or roll angle. This is undesirable because the vehicle's dynamics can be affected making it to be out of control. Thus, the Safety_Officer will cut off the thruster and abort the mission whenever high pitch or roll angle is detected to avoid the vehicle from going out of control.

Algorithm 1 shows how the vehicle and mission safety are checked and evaluated

in Safety_Officer component. To ensure the safety of the vehicle, the component implements an infinite loop for safety checking as long as the vehicle is turned-on.

Algorithm 1 Vehicle Safety Check

Require: Health_Monitor and Sentuator.

```
1:  loop
2:      for all Health Records in Health_Monitor do
3:          check HEALTH and SEVERITY
4:          if HEALTH ≠ HEALTHY and SEVERITY == ABORT then
5:              Inform Captain
6:          else if HEALTH ≠ HEALTHY and SEVERITY == EMERGENCY
            then
7:              Drop Ballast and Cut Vehicle Power
8:          end if
9:      end for
10:     for all Safety Issues do
11:         overLimit = checkLimits(Sentuator Data)
12:         if overLimit == true then
13:             Abort Mission
14:         end if
15:     end for
16: end loop
```

More safety issues can be defined in the future for multi-AUV mission operations. These include minimum or maximum distance among the AUVs and mission area boundaries.

3.4 Mission Level

Mission level concerns with the translation of a given mission file into individual tasks and disseminates them to the components at the vehicle level for mission execution. There is only one component at this level: the Task_Planner. Task_Planner retrieves

tasks from mission file, plans the task sequence and outputs the task commands as well as mission path for the mission execution.

3.4.1 Task_Planner

Whenever a START command is received from the Captain, the Task_planner reads the mission file to retrieve the task sequence, the mission parameters as well as the mission points. The retrieved mission points are then fed to the path planner for mission path planning. If a feasible path is found between the start and target mission points, the resultant tasks are passed to vehicle level for navigation, otherwise, Captain is informed and the mission is aborted. The following describes the path planner used for mission path generation, the format of mission file used as well as the translation of a mission into individual tasks.

Mission Script and Mission Task

In STARFISH project, missions are written in XML format and is called mission script. XML has been used widely as an universal data description language. It has been applied successfully in robotic applications as a tool for system integration as well as agent communication [15, 14]. Its structured well-formed document format is suitable to specify the mission points in sequential order as well as desired mission behavior in optional sub-tags.

A mission script can contain one or more missions identified by the mission number. Each mission consists of a group of default mission parameters and a set of mission legs arranged in sequential order. Every mission leg defines a high level mission task denoted by the keyword *type* and their corresponding 3D mission point denoted by x, y and z position. The default mission parameters can be overwritten

```
<?xml version="1.0"?>
<missions>
  <mission no="1" TimeOut="60">
    <parameters CruisingSpeed="1" SafetyDistance="2.5" WaypointRadius="10"
      MaximumDepth="50" MinimumAltitude="1" />
    <leg type="STEER" x="-399" y="947" z="-5" />
    <leg type="STATIONKEEPING" duration="100" x="-399" y="947" z="-5" />
    <leg type="STEER" x="-52" y="1350" z="-13" >
      <parameter CruisingSpeed="1" />
    </leg>
    <leg type="STEER" x="144" y="1643" z="-10" />
    <leg type="STEER" x="577" y="1724" z="-1" />
  </mission>
  <mission no="2" TimeOut="60">
    <parameters CruisingSpeed="0.7" SafetyDistance="2.5" WaypointRadius="10"
      MaximumDepth="50" MinimumAltitude="4" />
    <leg type="STEER" x="200" y="400" z="-3" >
    <leg type="STEER" x="400" y="600" z="-3" >
      <payload device="SIDESONARSCAN" duration="10"/>
    </leg>
    <leg type="STATIONKEEPING" x="400" y="600" z="-3" />
    <leg type="STEER" x="100" y="100" z="0" />
  </mission>
</missions>
```

Table 3.1: sample XML mission file.

if necessary by adding the *parameter* sub-tag within the mission leg tag. Whenever payload sections are attached, they can be turned on and off to obtain data by adding the *payload* sub-tag within the desired mission leg. This allows parallel execution of multiple devices on the AUV. An example of XML mission file is shown in Table 3.1 and its mission Document Type Definition (DTD) is shown in Table 3.2.

From the user's perspective, a STARFISH mission consist of a set of mission tasks with optional sub-mission parameters and payload parameters. Besides being the mission points for navigational pattern, the position tuple (*x,y,z*) that attached with each mission leg denotes the end point for the execution of particular mission task. For instance, a triple (*x=10,y=50,z=-5*) with *type="STEER"* and payload sub-tag *device="SIDESCAN"* informs the AUV to turn on the SIDESCAN sonar while navigating from the current position to the point (10,50,-5).

```
<?xml version="1.0"?>
<!DOCTYPE missions [
  <!ELEMENT missions (mission*)>
  <!ELEMENT mission (parameters, leg+)>
  <!ATTLIST mission no ID #REQUIRED>
  <!ELEMENT parameters EMPTY>
  <!ATTLIST parameters CruisingSpeed CDATA #REQUIRED
                       SafetyDistance CDATA #REQUIRED
                       WaypointRadius CDATA #REQUIRED
                       MaximumDepth CDATA #REQUIRED
                       MinimumAltitude CDATA #REQUIRED>
  <!ELEMENT leg (parameter?, payload*)>
  <!ATTLIST leg type CDATA #REQUIRED
                x CDATA #REQUIRED
                y CDATA #REQUIRED
                z CDATA #REQUIRED>
  <!ELEMENT parameter EMPTY>
  <!ATTLIST parameter CruisingSpeed CDATA #IMPLIED
                      SafetyDistance CDATA #IMPLIED
                      WaypointRadius CDATA #IMPLIED
                      MaximumDepth CDATA #IMPLIED
                      MinimumAltitude CDATA #IMPLIED>
  <!ELEMENT payload EMPTY>
  <!ATTLIST payload device CDATA #REQUIRED
                    duration CDATA #IMPLIED>
]>
```

Table 3.2: DTD for XML mission file.

Whenever a command is received to start a mission, the Task_Planner loads and processes the XML mission script based on the specified mission number. The mission points are retrieved for path planning while the mission tasks are converted to a mission array and optional parameters are recorded if they exist. Mission tasks is a set of high level sequencing task information. Each task compose of one or more low level vehicle behaviors, which executed together in parallel to generate the desired navigational pattern specified by the mission task. However, the breaking down of mission task to vehicle behaviors is the responsibility of the vehicle level.

To provide better explanation of the translation from mission file to mission task sequence, an example is shown in Fig. 3.2. This translation of mission is based on the mission No. 2 of mission file shown in Table 3.1. From the figure, it is seen

that all the mission legs from the mission file are retrieved and store in sequential order together with payload section. When the mission starts, the mission sequence informs the vehicle to traverse to (200,400) and at a depth of 3 meter below the surface. Once there, it performs a survey mission with Side Sonar Scan (payload) while traversing to (400,600) at the same depth. Once the second mission point is reached, it performs station keeping at that same position until a continue command is received from the operator (since there is no duration specified for station keeping). When the command from operator is received, it surfaces at (100,100). Throughout the mission execution, safety limits of minimum altitude = 4 m and maximum depth = 50 is imposed. The whole mission is aborted if it is still running after 60 minutes or any of the safety limits are violated.

Path Planning

Mission path is a set of three dimensional waypoints that define the AUV's navigational pattern to fulfill the mission requirements. A safe path not only is crucial to ensure the AUV's safety, but also affects the mission outcome. During the planning process, path planner takes the mission points from the script and tries to find a safe yet shortest path between the vehicle position and the target. If a feasible path is found, waypoints are generated and sent to vehicle level for navigation.

The existence of unpredictable sea current and obstacles makes path planning in three dimensional partially unknown environment a challenging problem. Many methods have been developed to tackle the problem. [25, 21, 2, 29] As part of this project, a novel particle based path planning algorithm [22]. The particle based path planning algorithm is developed based on particles generation and evaluation. Random particles are generated at every planning step starting from the origin to the

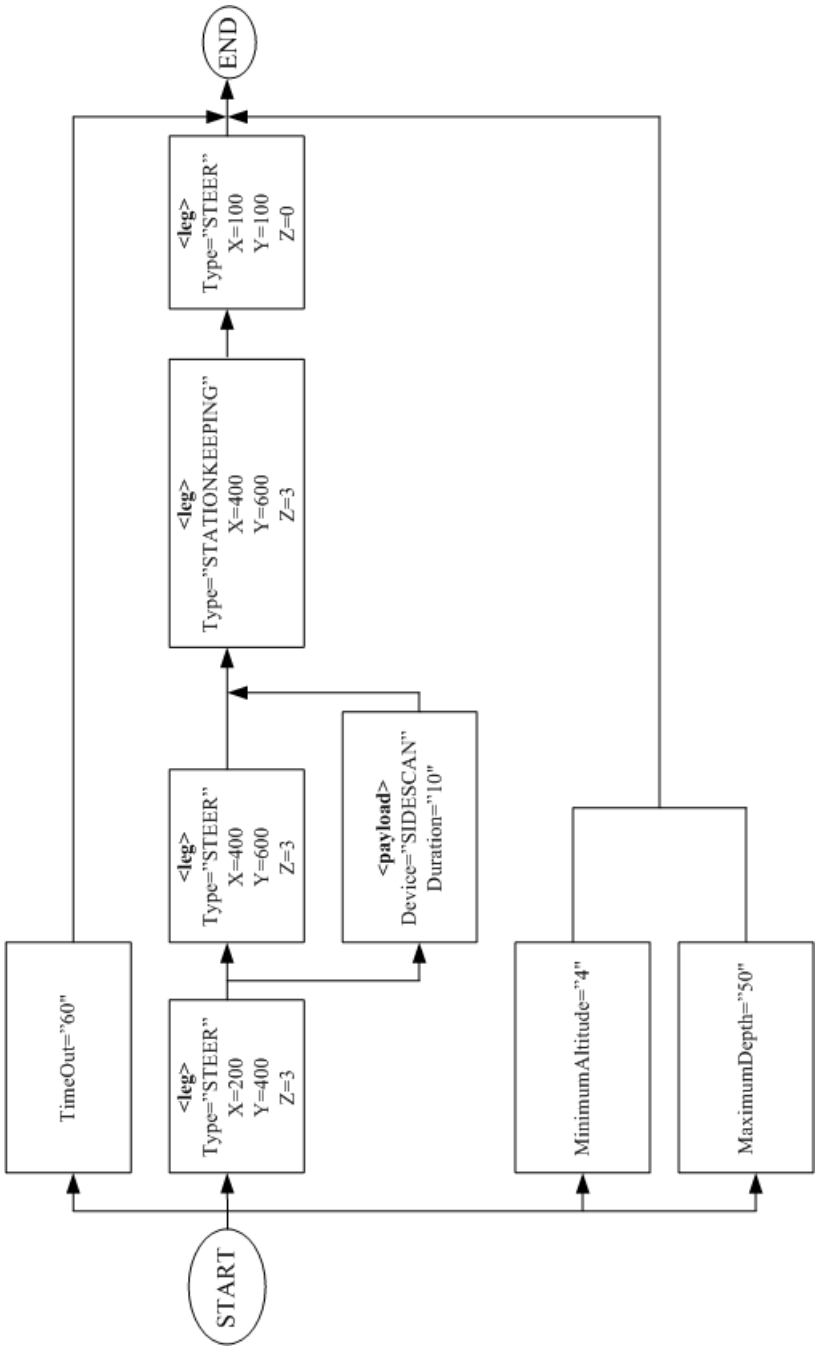


Figure 3.2: Translated mission tasks sequence based on mission file.

target. The particles are then assigned with different weights and evaluated according to the following principles :

1. Moving towards the goal (the smaller the distance between the ith particle's position and the target position, the larger the ith weight);

2. Avoiding obstacles (if the ith particle drops in the obstacle area, its corresponding weight is set to zero);

3. The size of sampling area of the particles is proportional to the vehicle turning and pitching angles (the area for particles generation is restricted by the vehicle's turning and pitching limitations);

4. Preventing the AUV from getting trapped in local minima (the larger the position difference of the ith particle, the larger its weight is. Position difference of a particle is the product of all the distances from the M previous planned waypoints).

Every particle has its velocity and position elements. The velocity element is obtained based on a 3D coordinate system with X, Y, Z axis and origin point O (Fig. 3.3).

A pair of angles, (θ, φ) is used to represent the direction of each particles's velocity in 3D space. The particle's velocity element is calculated by:

$$\begin{aligned} v_k^i = & \\ & ||v_k|| \times \{cos(\alpha_{k-1} + \gamma_k^i), sin(\alpha_{k-1} + \gamma_k^i), \\ & cos(\alpha_{k-1} + \gamma_k^i) \cdot sin(\alpha_{k-1} + \gamma_k^i) \cdot tan(\beta_{k-1} + \chi_k^i)\} \end{aligned} \tag{3.4.1}$$

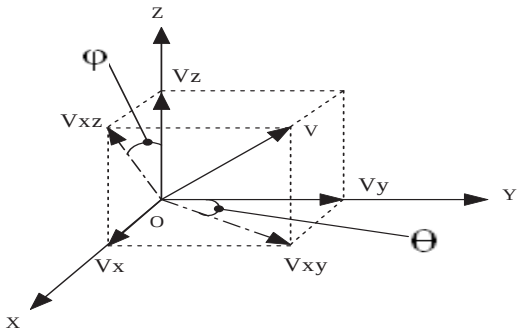


Figure 3.3: Three Dimensional (3D) coordinate system.

where α_{k-1} and β_{k-1} are respectively the AUV's bearing angle and pitching angle at time step k-1 (Principle 3). Random angles γ_k^i and χ_k^i are respectively generated from the uniform distributions and can be limited by AUV's bearing and pitching constraint.

On the other hand, the position element is obtained according to the following point mass motion model,

$$s_{k+1}^i = l_k + v_k^i \cdot T, \tag{3.4.2}$$

where l_k is the AUV's position at time step k (the kth planned waypoint) and T is the time interval. In order for the AUV to escape from a local minima (Principle 4), a large position difference is preferred and considered in the evaluation of the weights of the particles (hypothesis). Position difference is defined as the sum of distance from current particle's position to the M number of previous planned waypoints and is calculated by :

$$\rho_{k+1}^i = ||s_{k+1}^i - l_{k-M+1}|| \times ||s_{k+1}^i - l_{k-M+2}|| \cdots ||s_{k+1}^i - l_{k-1}|| \times ||s_{k+1}^i - l_k||. \tag{3.4.3}$$

ρ_{k+1}^i is chosen as the criterion to evaluate the ith particle and is proportional to its weight.

Once the evaluation is completed, the particle with the highest weight is chosen as the next waypoint for the mission path. This process will be repeated for every planning step until the target point is reached. The proposed path planning algorithm provides robustness and flexibilities as it allows different parameters to be defined according to the vehicle dynamics. Besides that, by considering only the sampling area within a world model and storing only the waypoints planned at every step, the computation time and memory requirement for path planning is greatly reduced.

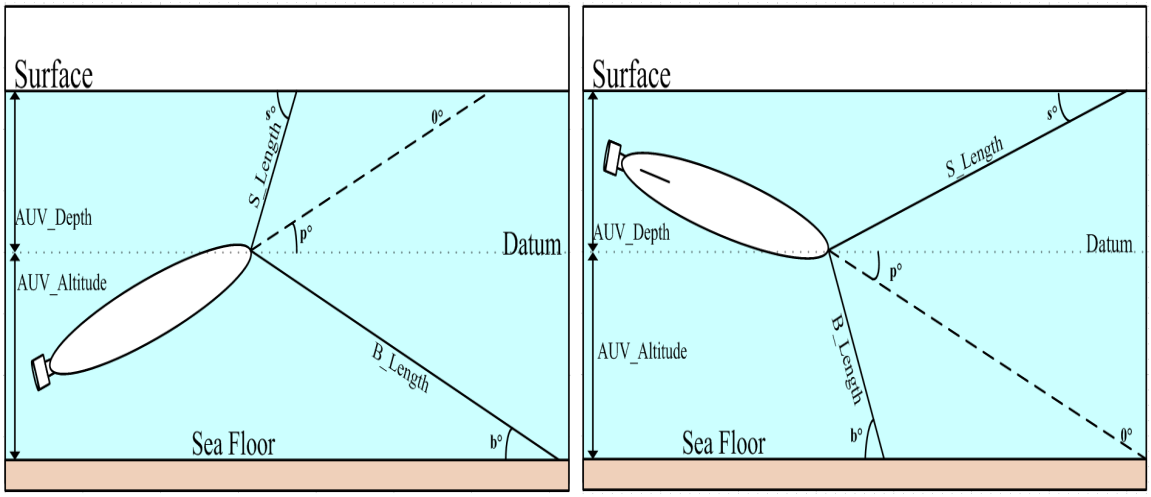
3.5 Vehicle Level

The Vehicle level consists of five components: Path_Executor, Obstacle_Detector, Chart_Checker, Scientist and Health_ Monitor. They are the reactive components that interact directly with the vehicle's sensory and actuator level - the Sentuator level. The control and processing at this level is distributed among the components. Every component has its own responsibility and operates asynchronously based on the commands instructed from the higher level of control hierarchy. Among the components at this level, the Path_Executor, Obstacle_Detector and the Chart_Checker together play the role of a pilot for navigation. They handle tasks ranging from translation of mission tasks into control signal, depth and position keeping, obstacle avoidance and mission chart updating.

3.5.1 Obstacle_Detector

During mission execution, floating obstacles and sea floor are threats to the AUV's safety. Collision with any threats may jeopardize the mission as well as the AUV. Early detection of unknown obstacles lying along the AUV's path is crucial to make

sure it has enough time and space to perform the avoidance maneuver. The Obstacle_Detector reads the data from the Forward Looking Sonar (FLS) that is mounted in the nose section of the AUV to determine the location of objects that exist within the sonar scan area. In order to do that, the raw data from the FLS need to be processed and the processing requires two separate steps: sensor noise filtering and obstacle detection.



(a) (b)

Figure 3.4: During mission execution, the AUV will either have positive or negative pitch angle. The sign of the pitch angle results in different calculation of the expected length of sonar beams. (a)negative pitch angle. (b)positive pitch angle.

Before any possible obstacles can be determined, the sensor noise (sea bottom and surface) from the sonar must be filtered. This is done by utilizing the depth and altitude data from the sensors to estimate the sea depth and surface altitude based on the AUV's current position (Fig. 3.4). Any sonar beams returned with their length fall within these range is considered as reflection from the sea bottom or surface and is filtered out. The calculation of expected beam length reflected from the surface (S_Length) and reflected from the sea bed (B_Length) depends on the AUV's pitch

angle. Since the FLS sends out sonar beams in a cone shape as shown in Fig. 3.4, the S_Length will be longer than the B_Length when the AUV has a positive pitch angle. The opposite is true when the AUV has negative pitch. The FLS sends out multiple vertical beams with 35° height and 3° width each. It is programmed to scan a sector of 90° horizontally. The calculations of both the S_Length and B_Length for both positive and negative pitch angles are as per 3.5.1 and 3.5.2. It is assumed that the FLS is mounted such that both S_Length and B_Length make an angle of 17.5° to the datum when the pitch angle of the AUV is 0°.

$$s^\circ = 17.5^\circ - \phi^\circ, \tag{3.5.1}$$

$$b^\circ = 17.5^\circ + \phi^\circ, \tag{3.5.2}$$

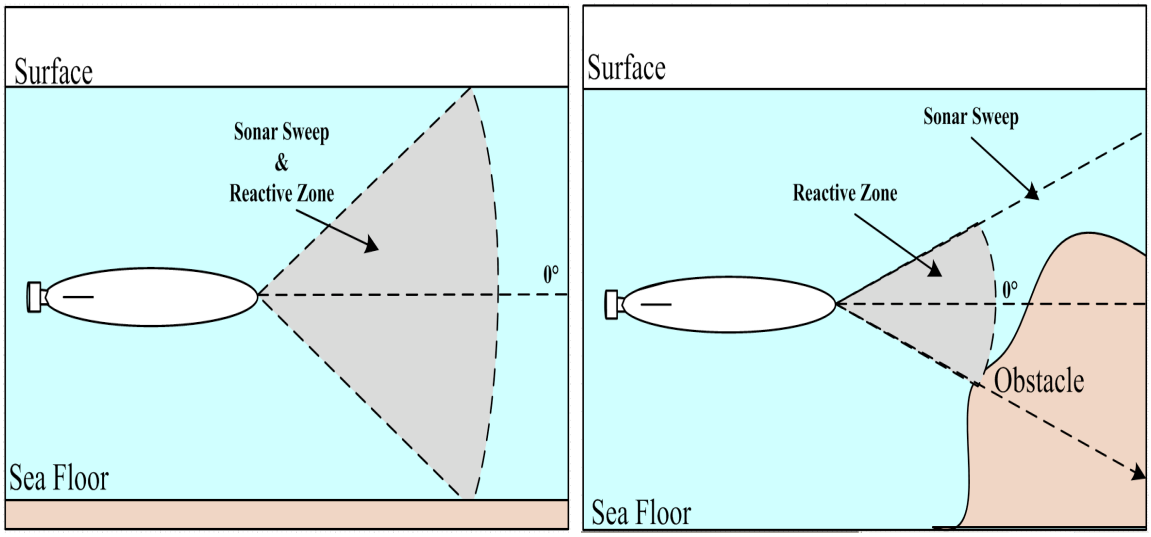
where ϕ is the pitch angle of the AUV, s° is the angle made by S_Length with the surface and b° is the angle made by B_Length with the sea bed. Since the measurement returned by Altimeter (Altimeter_Altitude) is based on the right-angle distance between the AUV's tail-nose axis with the sea bed, it will not depict the AUV's exact altitude whenever the AUV has a pitch angle. However, the perpendicular distance of the AUV to the sea bed (AUV_Altitude) can be calculated by:

$$AUV\_Altitude = Altimeter\_Altitude \times \cos(\phi). \tag{3.5.3}$$

Once the angles and AUV_Altitude are obtained, both the S_Length and B_Length can be calculated with :

$$S\_Length = \frac{AUV\_Depth}{\sin(s^\circ)}, \tag{3.5.4}$$

$$B\_Length = \frac{AUV\_Altitude}{\sin(b^{\circ})} = \frac{Altimeter\_Altitude \times \cos(\phi)}{\sin(b^{\circ})}. \quad (3.5.5)$$



(a) (b)

Figure 3.5: Forward Looking Sonar model vertical view. (a)When no obstacle exist in the sonar sweep, the reactive zone is the largest until it is reflected from the sea surface/floor. (b)When obstacle presents, the reactive zone will be smaller depends on the range of the detected obstacle from the AUV.

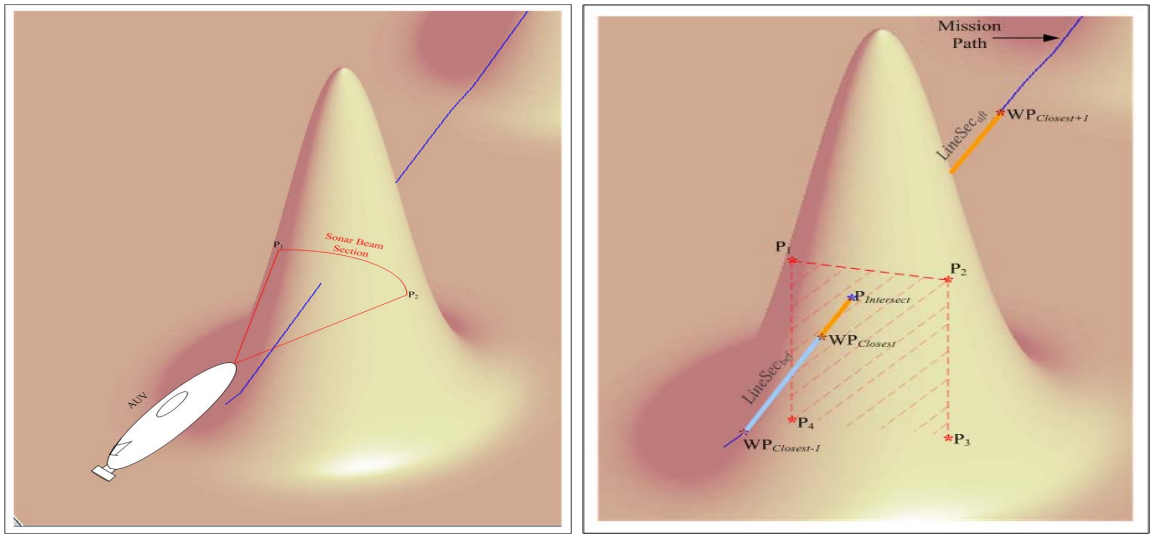
With that, any measurement returned by the FLS that is less than both the S_Length and B_Length is considered as reflection from the objects located within the cross-section of the sonar coverage and is used for obstacle detection. Fig. 3.5 shows the process of obstacle detection. The location of detected obstacle is calculated as follows:

$$\begin{bmatrix} X_o \\ Y_o \\ Z_o \end{bmatrix} = \begin{bmatrix} X_{AUV} \\ Y_{AUV} \\ Z_{AUV} \end{bmatrix} + B_r \begin{bmatrix} cos(W)cos(Q) \\ sin(W)cos(Q) \\ sin(Q) \end{bmatrix}, \tag{3.5.6}$$

where X_o, Y_o and Z_o are the 3D components of Obstacle's location; X_{AUV}, Y_{AUV} and Z_{AUV} are the 3D components of AUV's location; B_r is the range of the sonar beam returned by FLS; W is the horizontal angle of the beam from AUV's tail-nose axis and Q is the vertical angle below datum. The effective range (reactive zone in Fig. 3.5-(b)) of FLS is dependent on the AUV's altitude as well as the mission site's maximum depth. The closer the AUV is to the surface and the shallower the mission site, the lower the FLS's effective range. Although this simple method is less efficient, it only requires low cost FLS which allows fast processing of sonar signal.

3.5.2 Chart_Checker

After the sensor noises are filtered out and the location of the obstacles are calculated, the processed data is sent to the Chart_Checker component which in turn verify wether the detected obstacle is present in the Chart Room. ChartRoom is the central storage for the maps of the mission areas where the AUVs are operating. Each map has two layers; the first layer has two dimensional arrays with numerical values specifying the depth at the corresponding coordinates, while the second layer records the forbidden zone where the AUV must not pass through during navigation. Obstacles known in the Chart Room are taken into account when planning mission path. Obstacles that do not exist in the Chart Room is marked and the corresponding location and depth are updated in the maps. Collision checking is then performed by the Chart_Checker component along the mission path. If any of the newly found obstacles lie in the mission path, Chart_Checker component forward the data to Path_Executor



(a) (b)

Figure 3.6: Collision Detection performed by Chart_Checker. (a)The AUV's sonar beam section when it is reflected by obstacles. (b)Two points on the edges of the sonar beam section (P_1,P_2) are considered for collision detection. The $LineSec_{bef}$ and $LineSec_{aft}$ are calculated before $P_{Intersect}$ is determined.

component for obstacle avoidance.

The process of collision detection between the newly detected obstacle and the mission path are shown as follows:

1. The obstacle data sent by Obstacle_Detector component is in the form of an array. Each entry in the array denotes a processed data from sonar beam which consists of the estimated location the sonar beam's intersection point with the obstacle, the distance of the obstacle from the vehicle and the sonar beam's angle with the vehicle's tail-nose axis. For simplicity, only the location of two beams from the widest edges of the sonar cross section is considered. These two points (P_1,P_2) form a straight line and is used to find the closest waypoint ($WP_{closest}$) in the mission path (MP). This is done by computing the shortest

distance from each of the waypoint in the mission path to that straight line.

2. The second step is to find the point of intersection($P_{Intersect}$), if there is any, between the mission path and the plane formed by the obstacle data. Two mission path segments immediately before ($LineSec_{bef}$) and after ($LineSec_{aft}$) $WP_{closest}$ are taken for intersection check. To form a plane, two extra points (P_3,P_4) are taken directly below P_1 and P_2 with identical X and Y positions but lower Z position. If there is no intersection point found ($P_{Intersect} = NULL$), the mission path does not collide with the detected obstacle and the mission continues.

3. Otherwise, further checking is needed to determine if the intersection point lies on the interior of the obstacle region. It is required to check wether the intersection point (collision point, $P_{Intersect}$) is inside the obstacle region enclosed by P_1, P_2, P_3 and P_4. The checking is done by computing the sum of the angles between the $P_{Intersect}$ and every pair of the edge points of the obstacle region. The sum of angles will be 2π if the point is on the interior of the obstacle region. Whenever the $P_{Intersect}$ falls in the obstacle region, the mission path that is within this vicinity is modified according the obstacle avoidance mode. This is the task of Path_Executor component. Detailed implementation of the collision detection is illustrated in Fig. 3.6 and Algorithm 2 at the end of the chapter.

3.5.3 Path_Executor

The Path_Executor is responsible for translating the high level mission tasks into vehicle's low level maneuver control. The component implements a library of basic

functions that the vehicle can use to generate the desired maneuver behaviors. One or more basic functions can be invoked concurrently to achieve a high level mission task. This is a bottom-up approach where distributed simple vehicle behaviors are merged to exhibit complex maneuver. Currently, there are three basic functions implemented in STARFISH project to fulfill the mission's requirements: *SteerToXY, MaintainPosition* and *ObstacleAvoidance*. Each basic function takes inputs from the sensory modules (Senstuator level) and output the control signal to the actuator modules independently. The selection of basic functions to be performed depends on the mission's current task as well as the state/activity of the component's state machine.

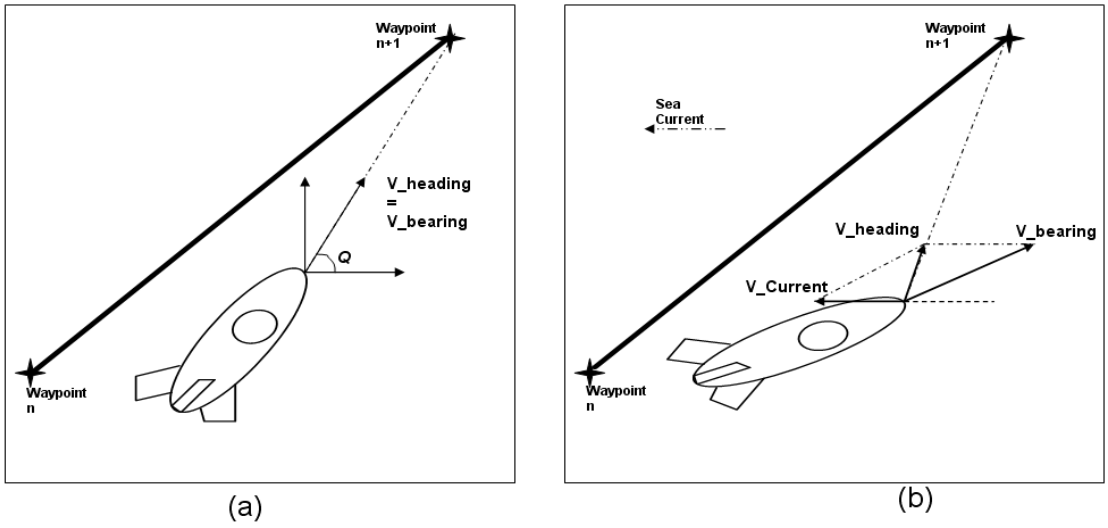


Figure 3.7: Path Following navigation. (a)Navigation without current. (b)Navigation with current

- ***SteerToXY:*** steers the vehicle from current position to given mission point in two dimensional plane, (x,y). This function takes in the current AUV's bearing signal and outputs the bearing offset to adjust the AUV's heading to the target

point. Bearing is defined as the direction in which the AUV's tail-nose axis is pointing while heading is the direction of the actual AUV's motion over time. A boolean function *PointReached(p)* is used to determine wether the AUV has reached the target point or not. When the distance between the AUV's current position and the target point is less than the *WaypointRadius* specified in the mission script, the boolean parameter is set to p=*TRUE* and new target point from the path is loaded to continue the navigation. This process is repeated until the AUV reaches the final mission point. Whenever sea current exist during AUV's operation, the bearing control is affected. In order to achieve path following behavior, the velocity of sea current is first estimated as follows:

$$V\_current = V\_heading_{current} - V\_bearing_{current}, \tag{3.5.7}$$

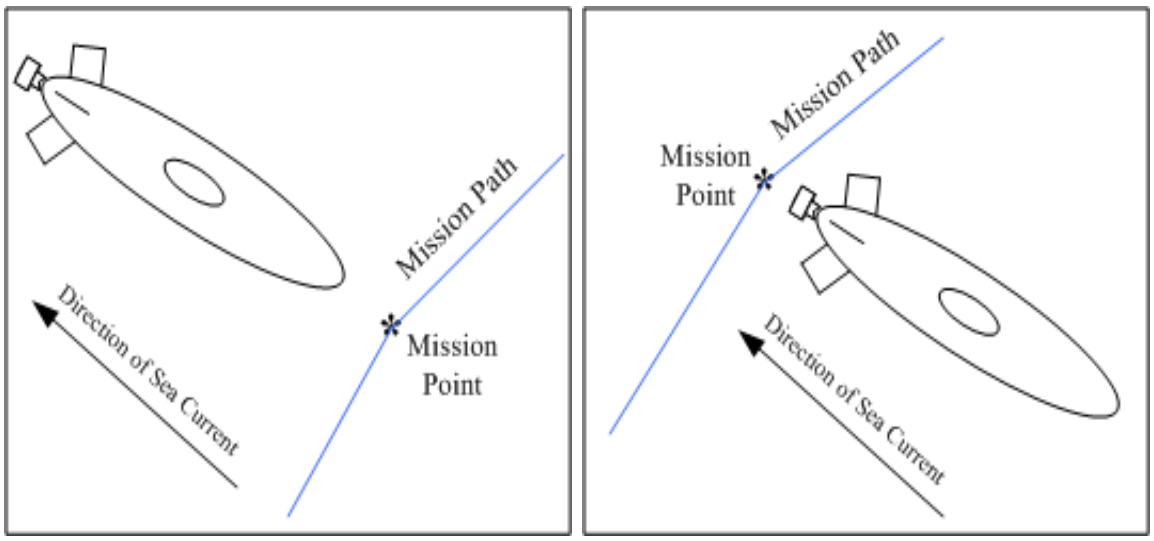
where *V_current* is the sea current velocity, $V\_heading_{current}$ is the AUV's current heading velocity and $V\_bearing_{current}$ is the AUV's current bearing velocity. The $V\_heading_{current}$ is obtained from Doppler Velocity Log (DVL) and Inertial Measurement Unit (IMU) while the $V\_bearing_{current}$ is estimated based on combining of thrust based computation of $\mid$ *V_bearing* $\mid$ and compass information of angle(*V_bearing*). Once the sea current velocity is obtained, the AUV's setpoint bearing is easily calculated:

$$V\_bearing_{new} = V\_heading_{desired} - V\_current. \tag{3.5.8}$$

Fig.3.7 illustrates the AUV's navigation with and without sea currents. The implementation of navigation with sea current compensation using constant thrust and varying bearing at the reactive level minimizes the control signal delay resulting in better path following behavior.

- ***MantainPosition:*** Although STARFISH AUV has slightly positive buoyancy and its design does not allow hovering over a target point, some degree of station keeping is still desired. The behavior is particularly useful when the AUV is waiting for response or command from the operator, or path replanning is required during mission execution. Under the effect of the positive buoyancy and sea current's disturbance, the AUV will float to the surface or drift by the current if position maintaining behavior is not available. Two different *MantainPosition* modes have been implemented in this project: with current and without current. When there is no sea current, the AUV hovers in a circle with certain fins angle so that the resultant propulsive force is able to counter the positive buoyancy and keep it at a certain depth. On the other side, if sea current exists, the AUV performs forward thrust against the direction of sea current until its location is in front of the station keeping point (Fig. 3.8-(a)) before turning the thruster off. At that point, as long as the AUV's depth and location are within the allowable distance from the station keeping point, it lets the force of the sea current to push it backward until its location is behind the station keeping point before turning on the thruster again (Fig. 3.8-(b)). This forward and backward motion allow the AUV to maintain its depth around vicinity of the station keeping position.

- ***ObstacleAvoidance:*** Since this behavior shares the same vehicle control with *SteerToXY* behavior, only one behavior is activated at a time. Whenever obstacle data is received from the Chart_Checker component, this behavior is activated to avoid obstacles. The control is passed back to *SteerToXY* once the vehicle is out of danger zone. Two *ObstacleAvoidance* behaviors have been



(a) (b)

Figure 3.8: Station keeping when sea current exist. (a)AUV is considered in front of the station keeping mission point (with respect to the AUV's body frame). (b)AUV is behind the station keeping mission point (with respect to the AUV's body frame).

implemented to increase the robustness of the STARFISH AUV in mission execution: the *depth control* and the *altitude control* avoidance modes. When *depth control* mode is used, the avoidance behavior is performed by navigating the AUV towards the right or left depending on the obstacle's location. On the other hand, in *altitude control* avoidance mode, the obstacle avoidance is done by instructing the AUV to go over the obstacle by changing the depth while maintaining its original course in xy plane and its altitude with respect to the ground. Only sea bed obstacles are considered in this case.

3.5.4 Scientist

Scientist component is responsible for processing and analyzing the data obtained from payload sensors. To allow exchangeable payload sections, one Scientist component is built per payload section and loaded by the Chief_Scientist depending on the AUV setup. More than one Scientist components can exist in a single AUV if there are several payload sections attached. They are coordinated and controlled by the Chief_Scientist component throughout mission. Two payload sections are currently built for STARFISH project. They are the Advanced Navigational payload and Side Scan Sonar payload. The AUV configuration shown in Fig. 6.1 is the Advanced Navigational payload which houses a DVL for more accurate velocity and altitude updates.

3.5.5 Health_Monitor

Health_Monitor component keeps track of the health conditions for all the devices in the AUV including the optional payload section. This is particularly important to make sure that the AUV is in its optimum working condition. Every hardware interfacing component (Sentuator component) in the AUV implements an internal health monitoring method which checks the health status of the hardware it is attached to. This information is retrieved periodically by the Health_Monitor component for vehicle health status updating and forwarded to the Safety_Officer for further analysis.

Message Type	Data Fields	Characteristics
Status Message (MDAT_STATUS_ STARFISH)	Message mode, AUV current coordinate, bearing, heading, pitch, roll, altitude, thruster, velocity, battery usage, current mission leg, AUV temperature.	Periodic, send updates to the mother ship to inform the status of the AUV at a particular point of time.
Path Update (MDAT_CURRENT_ PATH)	Message modes, mission leg number, 5 waypoints from the current mission's path.	Periodic, allow the AUV to inform the mother ship regarding the mission's next 5 waypoints starting from the current position.
Command Message (MDAT_COMMAND)	Message mode, message ID, command number, the first waypoint to be changed in the current mission path and 4 new waypoints to be added into the current mission path (optional, only required if command is to modify the mission points).	Sent by the mother ship. Allows mother ship to send command to the AUV to execute commands such as start, stop, abort, or change mission point etc.

Table 3.3: Table summarizing the CCL messages used for AUV communication.

3.6 External Communication

3.6.1 Signalling_Officer

Communications with the mission operator/mothership or other AUVs is supported by the Signaling_Officer through a message-passing mechanism. The Signaling_Officer acts as the AUV's external communication node, and is represented outside the overall control hierarchy. This component deals with the encoding and decoding of the messages between AUVs or between AUV and the operator. Besides that, the Signaling_Officer is also responsible for updating the operator with the current mission

and AUV status periodically. As proposed in [12], The Common Control Language (CCL) is adopted as the message format for acoustic communication. CCL is developed to establish a standard for communication between different agents during operation and provide supports for interaction between machines and operators. A CCL message is a data packet with 32 bytes in length. The first byte is used to specify the message mode (type) followed by optional data values. The message is encoded into a string of hexadecimal values using twos-complement method to maximize the limited bandwidth of underwater communication. Although some precision will be lost during the encoding process, they are not significant to affect the overall operational performance of the AUV. Fig. 3.9 shows the communication mechanism between the mothership/operator and AUV by using CCL message. The two parties which involved in the communication requires a CCL interpreter in order to carry CCL message encoding and decoding. While the interpreter resides in the operator's terminal at the operator side, the interpreter in the AUV is implemented as part of the Signaling_Officer component.

At current stage of STARFISH project, three CCL messages are defined for the communication between the operator and the AUV (Table 3.3). The first CCL message is the Status Message, which updates the operator with the vehicle's current status like coordinate, bearing, heading pitch and roll etc. periodically. This is important so that the operator is aware of the AUV's condition when it is operational underwater. The second CCL message is the Path Update message, which sends the current mission leg number together with the next 5 waypoints from its current path. The waypoints are be plotted on the control GUI so that the operator can keep track of the AUV's navigational plan. The third CCL message is the Command Message.

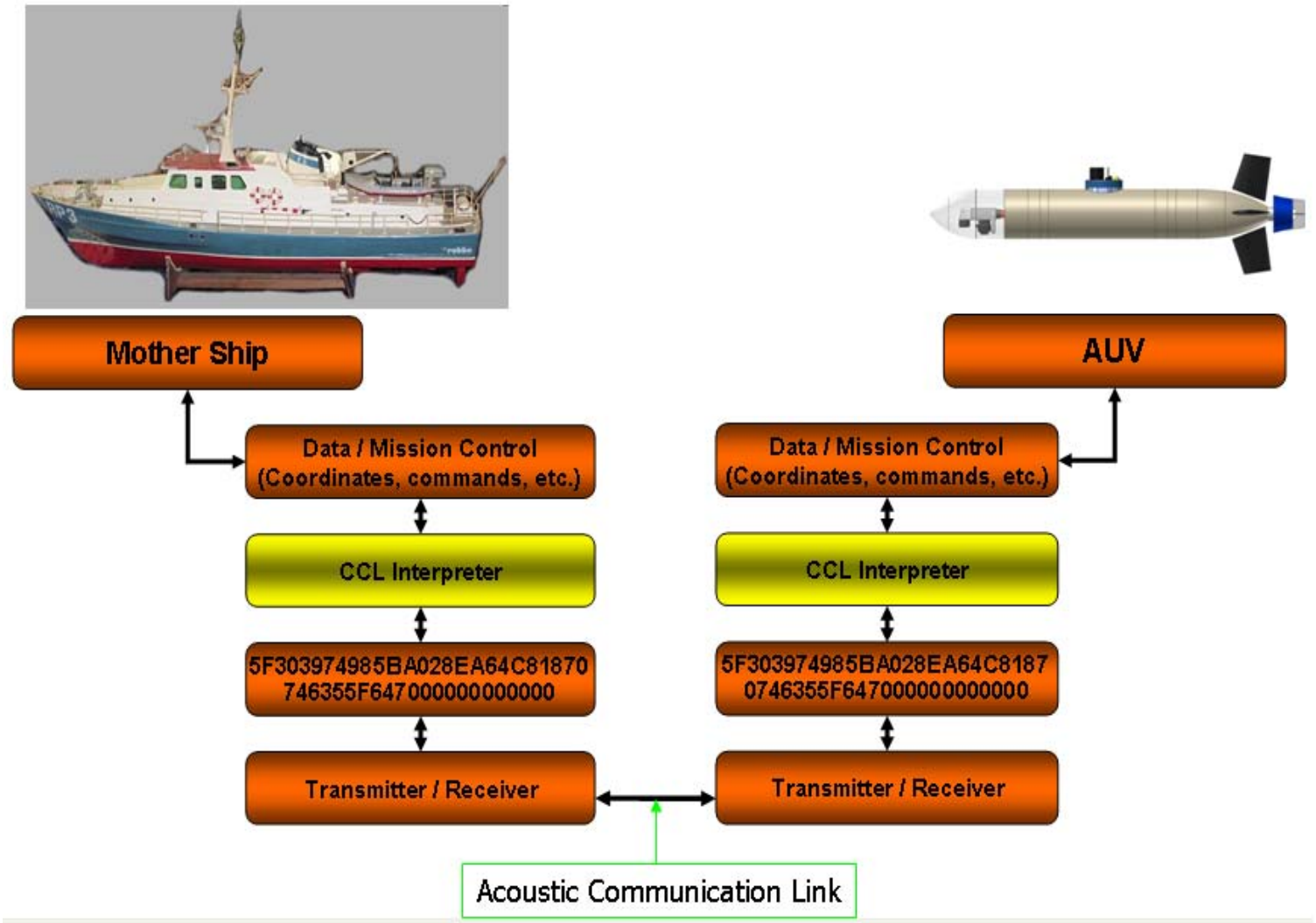
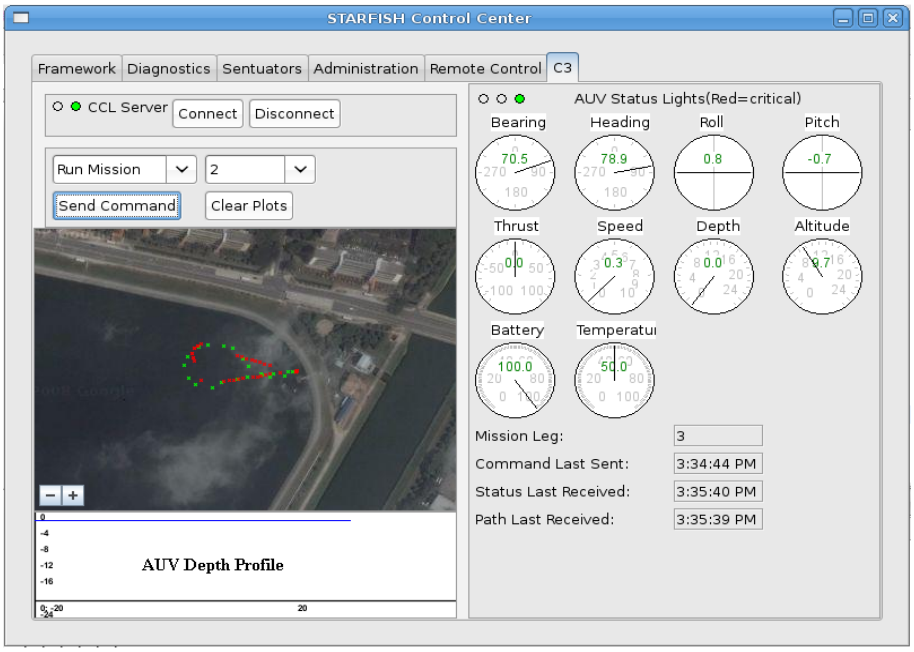


Figure 3.9: Communication mechanism between mothership/operator and AUV using CCL message.

This is the only message that is sent from the operator to the AUV. The message consist only the high level commands like mission start, mission stop, mission abort and mission change. In the case when mission change is requested, the remaining space in the message could carry up to 4 new waypoints if necessary.



Sample CCL: `#C,1,0,1,0,1,5ffffc000000003238ffb50057006100036432640300000000000000000000000`

Figure 3.10: Screen capture of the C2 GUI interface and a sample CCL message.

All three messages are currently being used for simulation test runs as well as lake test for single AUV operation. During mission execution, different CCL messages are sent periodically to the operator's machine for decoding and displayed via GUI interface. Fig 3.10 shows a sample CCL message string with corresponding representations and, the screen capture of Command and Control GUI module during operation. More CCL messages can be defined in the future for multi-AUV scenario.

3.7 Summary

In this chapter, the proposed C2 system architectural design is presented and illustrated in Fig. 3.1. The C2 system has a hybrid hierarchical control architecture where it demonstrates deliberative behavior at the high-level mission control while performing reactively for the low-level vehicle control. The control architecture is divided into three different levels where each level handles different mission tasks. The mission tasks in the C2 system are implemented into individual components. The components are arranged in hierarchical order within the three control levels to depict the different command responsibilities in the overall control system.

The components resides in the Supervisory Level deal with mission and vehicle monitoring while the components in the Mission level handles mission and tasks planning and finally, the components in the Vehicle level performs low-level vehicle maneuver control as well as obstacle detection and avoidance. The C2 system also equipped with an external communication component that is responsible for the communication between the operator and AUV, among the AUVs in a multi-AUV scenario.

Algorithm 2 Mission Path Collision Checking

Require: P_1, P_2, P_3, P_4, MP
Ensure: **true**→ collision, **false**→ no collision

1: **for all** WP *in* MP **do**
2: $Dist$ = getPointLineDist(WP, P_1, P_2)
3: **if** $Dist < Lowest$ **then**
4: $Dist = Lowest$
5: $WP_{closest} = WP$
6: **end if**
7: **end for**
8: $LineSec_{bef}$ = getLineSegmentBef($WP_{closest}$)
9: $LineSec_{aft}$ = getLineSegmentAft($WP_{closest}$)
10: $Obstacle_{plane}$ = getPlane(P_1, P_2, P_3)
11: $P_{Intersect}$ = getIntersection($Obstacle_{plane}$,$LineSec_{bef}$, $LineSec_{aft}$)
12: **if** $P_{Intersect} \neq$ NULL **then**
13: **for** $i = 0$ *to* 4 **do**
14: ϕ = getAngle($P_{Intersect}$, P_i, $P_{(i+1)\%4}$)
15: $angleSum$ += ϕ
16: **end for**
17: **if** $angleSum == 2\pi$ **then**
18: **return true**
19: **else**
20: **return false**
21: **end if**
22: **else**
23: **return false**
24: **end if**

Chapter 4

Command and Control Software Architecture

4.1 Introduction

This chapter presents the underlying software architecture for the command and control system. First, the overall software architectural design is depicted in Fig. 4.1. This is followed by the description of the software architecture's basic building blocks, the container and the component object in Section 4.3. Section 4.3.1 investigates further into the component's internal structure while Section 4.3.2 explains the components external communication interfaces. Finally, Section 4.4 describes how the component communicate with each other within the system.

4.2 Architectural Overview

The software architecture defines how the structure of different software components in the AUV are built and integrated to form a fully functional system. Fig. 4.1 shows the overall software architecture of the STARFISH AUV. The architecture is composed of a set of software components distributed among the onboard PC104+

and the MicroController Units (MCUs). All the components are connected by their own local software server and, the communication among the components is conducted through Ethernet switch with a common RPC based communication protocol throughout the AUV. More details of the software components construction and communication implementation is described in [9]. This design provides good overall software organization that helps in maintainability.

Like all other modules in the software architecture, the C2 system consists of individually built software components - the C2Component, that communicate over the same link and communication protocol. Besides that, the C2 system also has its own local software server - the C2Server, which serves as the central connection point for all the C2Components. Overall, the C2 system resides at the highest level and utilizes the services provided by the distributed software architecture described in [9].

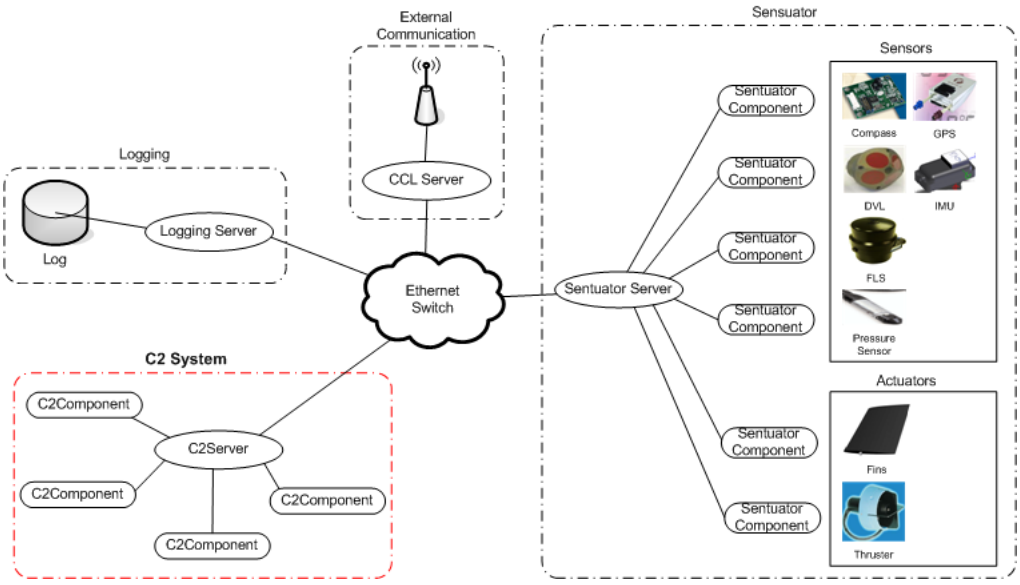


Figure 4.1: STARFISH AUV Software Architecture

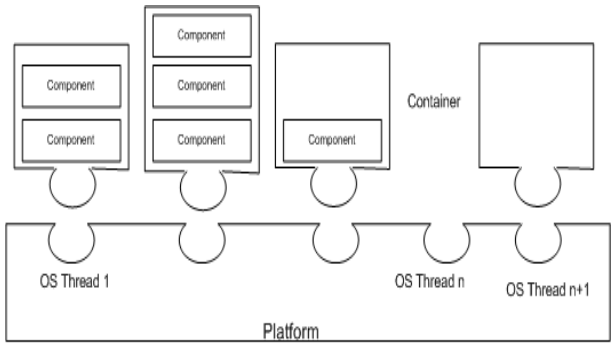


Figure 4.2: The software container together with its components. Each container defines a thread which runs at the platform.

4.3 Container and C2Component Object

The proposed C2 system software architecture is implemented as a multi-threaded program in the STARFISH AUV. It consists of multiple software containers where each container defines a thread in the operating platform. For simplicity, the Containers have been omitted in Fig. 4.1. However, the overall software architecture is not affected because the containers only concern the low level implementation on the operating platform.

Every Container implements a heart beat method, which is invoked by the operating platform in a regular basis. The heart beat (*tick* method) frequency can be adjusted to satisfy the system's hard and soft real-time requirements. A container can contain one or more components as shown in Fig. 4.2. The grouping of components into a particular container is based on the assigned responsibilities and behaviors under the C2 system. Each component is invoked sequentially by the associated container via its *tick* method to perform the assigned specific tasks. For example, the Task_Planner and the Path_Executor component are located in different containers because they carry out very distinctive command and control task in the C2 system.

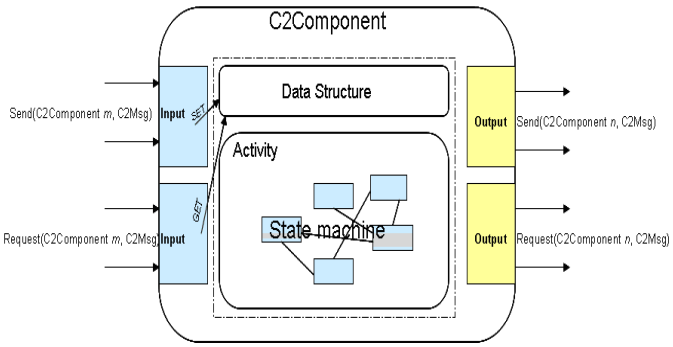


Figure 4.3: C2Component and its internal structure as well as external communication interface.

The Obstacle_detector and the Chart_Checker component are put into the same container because they deal with obstacle detection and avoidance. The Chart_Checker component has to wait for obstacle data from the Obstacle_Detector component when an obstacle is detected before executing any avoidance behavior.

In the proposed C2 system, command and control tasks are divided into individual components. The division is natural where each component is self-contained and the components perform assigned tasks. For example, the Task_Planner Component is responsible for task and path planning while the Safety_Officer Component monitors the AUV's condition to ensure its safety. Components are the basic functional units of the whole C2 system. There is a simple interface class C2Component (Fig. 4.3), any component that implemented this interface can be added into containers under the C2 system. The C2Component interface also implements a common component communication method to facilitate inter-component communication.

Fig. 4.4 shows the UML diagram of a 'container', a 'Component' and inherited 'C2Components'. One container can contain one or more C2Components, each C2Component in a container is invoked once sequentially by the Container's heart beat to perform its command and control tasks.

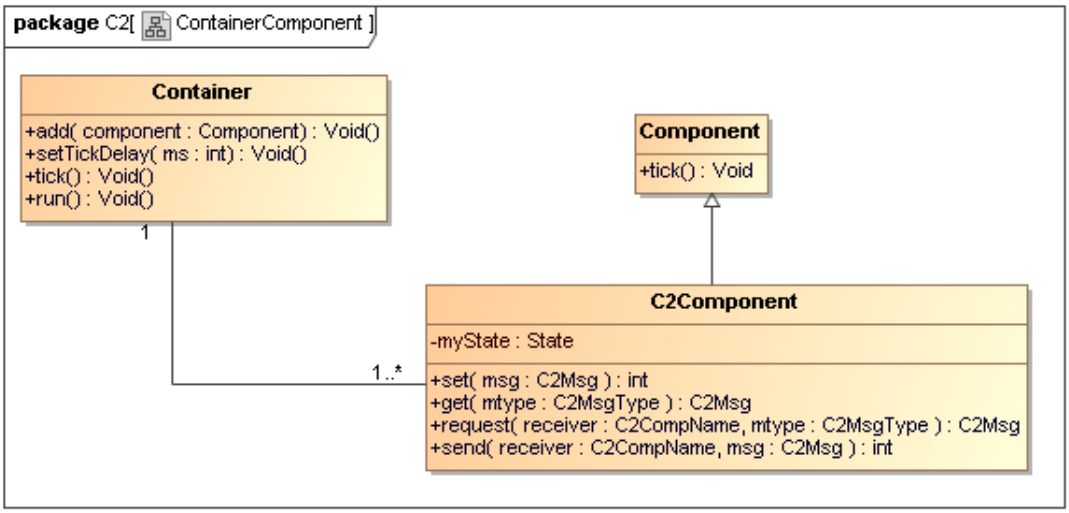


Figure 4.4: Container, Component and C2Component.

The following are two methods for C2Component's input and output channels:

- *Send* and *Request* methods as the output channel.
- *GET* and *SET* methods as the input channel.

Besides that a default State variable is used to keep track of the component's state. Every C2Component can define its own data structure and parameters depending on its tasks and responsibilities. They are private to the particular component and only accessible by the component's *GET* and *SET* methods. Activity is the main processing unit in a component; it implements the main algorithms that generate the desired behaviors or outputs for the component. However, the type of activities that are executed at a particular time is depending on the component's current state, which is controlled by a finite state machine. A *tick()* method is implemented at the component level too. When this method is invoked by the parent container, the finite state machine checks the components's current state, and executes the activities that

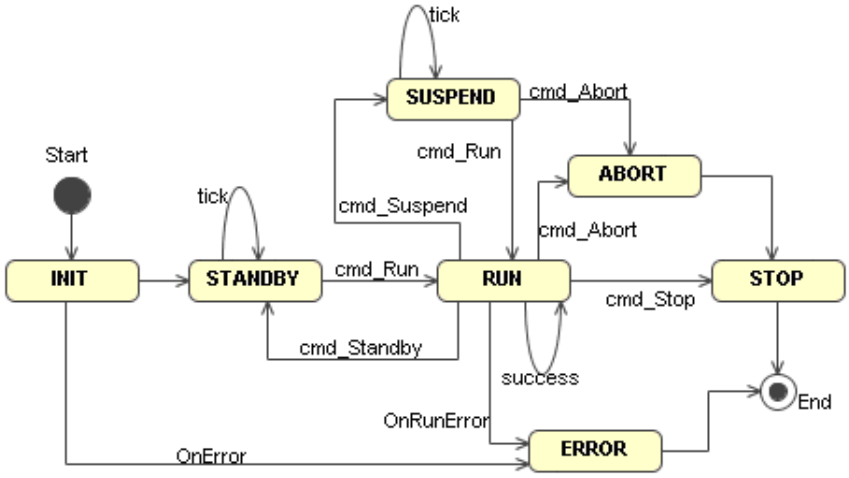


Figure 4.5: C2Component State Transition

are defined for that particular component state.

4.3.1 Activity and state transition

All C2Components have a finite state machine which processes their tasks continuously depending on the current state of the component. The transitions between states are triggered by commands from components at higher control hierarchy and/or component's internal events. The current state of a particular component can be monitored and controlled by another component in the same architecture. There are seven default states in all the C2Components: INIT, STANDBY, RUN, ERROR SUSPEND, ABORT and STOP. However, extra sub-states can be defined to handle more complicated tasks and algorithms in the component. Fig. 4.5 shows the state transition chart of C2Component.

The implementation of state machine in C2Components is to facilitate controllability and observability in the control architecture [11]. Both controllability and observability allow monitoring and controlling of the internal structure and behavior

of the C2Component. This is particularly important in a C2 system where supervisory components at the higher level control architecture can monitor and command the behavior of lower level components.

4.3.2 Component input and output methods

As a mean of communication with other components, all C2Components own two default output methods: *Send* and *Request*. It also provides two input methods: *SET* and *GET*. *Send* is invoked by a component to provide information to the target component, while *Request* is invoked to ask for information from the target component. When the *Send* method is invoked from the source component, the corresponding *SET* method is invoked in the target component to accept the information. On the other hand, *GET* method is invoked in the target component to provide information for the requesting component. Detailed operation of the interaction among the C2Components using the component's input and output methods is explained in the following section.

4.4 Component Communication mechanism

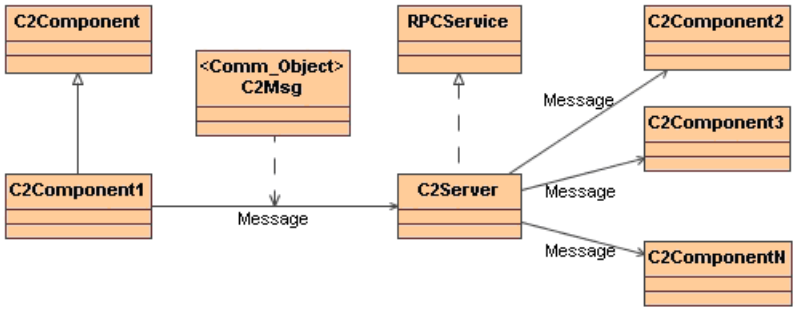


Figure 4.6: C2Component communication via C2Server and C2Msg.

The proposed C2 system provides a simple yet effective way for inter-component communication. As shown in Fig. 4.6, all the components communicate with each other by using C2Msg data structure through the input and output methods.

4.4.1 Communication Object : The C2Msg

C2Msg is a communication object which encapsulates the data types that can be exchanged among the components. This provides uniformity for communication among the components, and allows developers to implement their own logics and algorithms in new components as long as it complies to the common communication interface. If needed, new data types can be added easily to the communication object in future to handle more complex data.

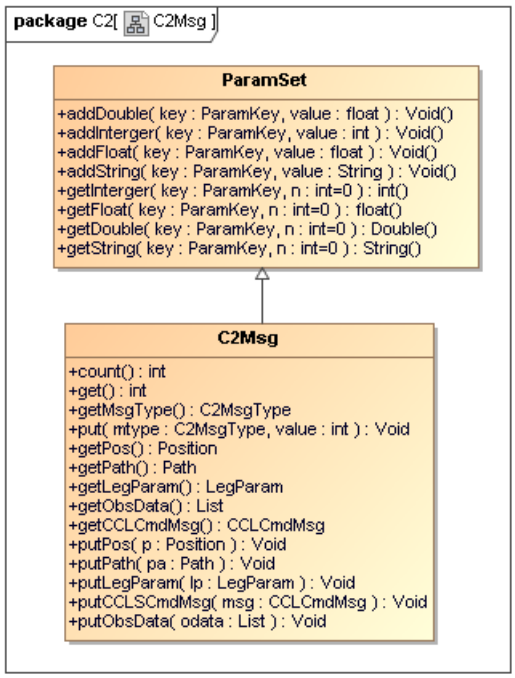


Figure 4.7: C2Msg class diagram.

Fig. 4.7 shows the UML diagram of the C2Msg class currently implemented in

the C2 system. At the current stage of the project, C2Msg can handle four complex data types in addition to the primitive data types that were provided by its based class - the ParamSet class. The four data types are handled by the methods described below:

getPos and putPos get or put Position data type which encapsulates *X, Y* and *Z* data fields to describe the location in a coordinate system. Each of the data field is a *Double* data type with both positive and negative value. Since the origin is a valid data during AUV's mission, *NULL* is used whenever the position is not available.

getPath and putPath get or put a path which consists of an array of Position data types. This is useful for the Task_Planner to pass the planned path to the Path_Executor and for the Chart_Checker to check collision between the mission path and detected obstacle. Whenever a path is not available, a *NULL* value is returned.

getCCLCmdMsg and putCCLCmdMsg get or put CCL command message. Apart from the mission start or abort commands, the command messages sent by the operator may consist other information like mission number and mission points to be changed in the current mission. Thus, these two methods provide an easy way for those data to be passed on to the designated C2Components.

getObsData and putObsData get and put obstacle data between the Obstacle_Detector and the Chart_Checker. The obstacle data consist of the obstacle's position, its angle from the AUV's current bearing and its distance from the AUV's current location. The obstacle data can be used in future to help the Captain in decision

making or for the Signaling_Officer to update the operator's terminal regarding the detected obstacles.

The complex data are packed into C2Msg when the put methods in sending component are invoked and returns the same message content when get methods are called in the receiving component. Although the number of data types are sufficient for the current version of C2 system, more data types can be added easily in the future if found necessary.

4.4.2 Point of Communication : The C2Server

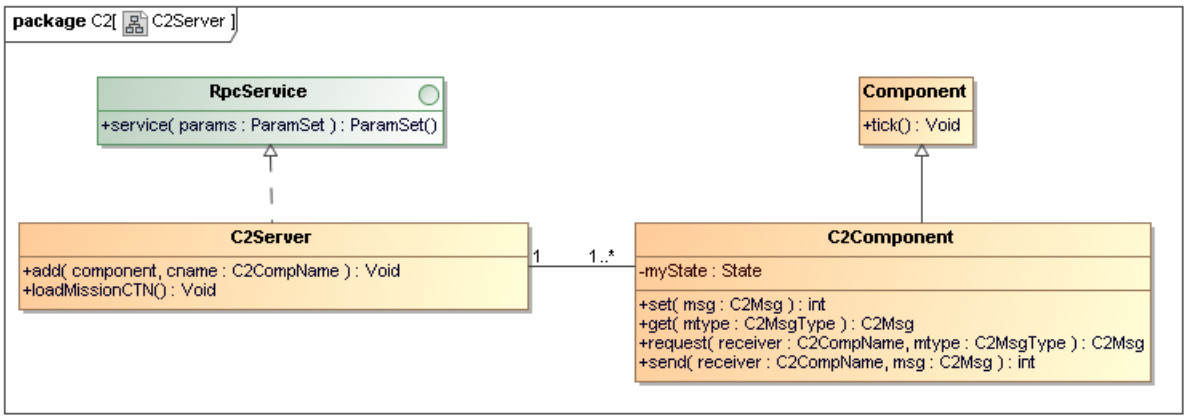
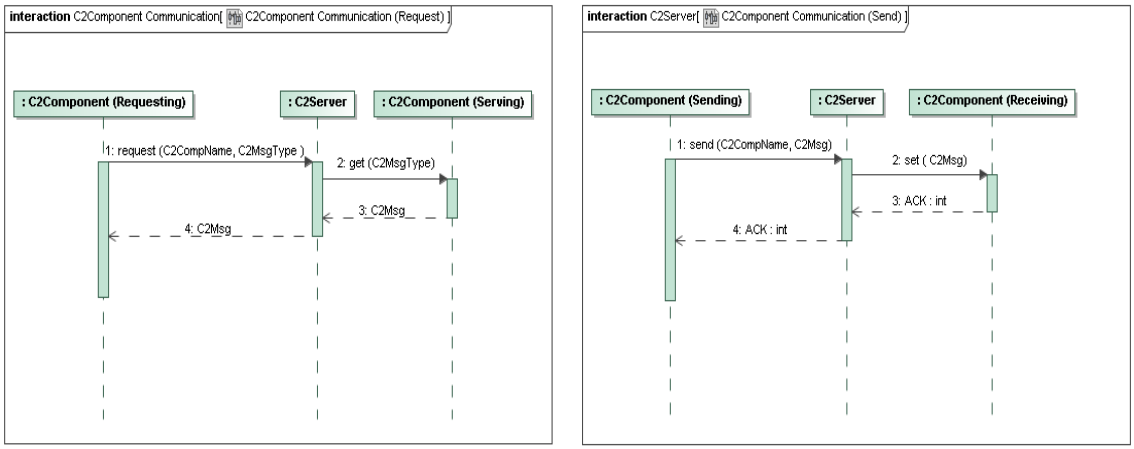


Figure 4.8: C2Sever class diagram.

Instead of connecting C2components directly to communicate with each other, they are connected to a central routing hub - C2Server as shown in Fig. 4.6. C2Server is responsible for delivering C2Msg from the source component to the destination component via Remote Procedural Call (RPC) service using a message passing protocol. Fig. 4.8 shows the UML diagram of C2Server and its relationship with the C2Components. Different from Container class, there is one and only one C2Server

for all the C3Components in the C2 system. All the C3Components have to be added to the server before they can interact with each other.

Besides being the central message routing node, the C2Server configures the C2 system when it starts. The *loadMissionCTN()* method reads all the Container/Component configuration from the mission file before adding them into the C2Server. Container/Component configuration specifies the container and C2Components to be loaded for the missions in the mission file. The operator can easily configure the C2Components to be loaded for a particular mission. This is very useful for the configuration of the Scientist components in a single AUV as well as for the configuration of AUVs in a multi-AUV scenario. The C2 system of the AUVs is equipped with all the Application Programming Interfaces (API) and only relevant APIs will be loaded depending on the individual AUV's hardware setup.



(a) (b)

Figure 4.9: Component Communication with C2Component and C2Server. (a)Sequence diagram showing a request operation. (b)Sequence diagram showing send operation.

Fig. 4.9 shows the sequence diagrams for both C2Components' request and send

operations to further illustrate the overall operation of the C2Component communication with C2Server. Whenever a C2Component needs information from another C2Component, a request operation is called with the serving component's name as well as the request message type. The information is packed as C2Msg and specified by the message type (C2MsgType). Upon receiving the request, the C2Server calls the get method of the serving component to obtain the requested information. If the information is available, it is returned in the form of C2Msg to the C2Server who forwards the message to the requesting component.

When a C2Component wants to send information to another component, it invokes the send method with the receiving component's name as well as the information packed as C2Msg. The C2Server then calls the set method of the receiving component to pass the information. Note that the C2Server only concerned wether the message is received or not by the component which is denoted by an acknowledgement (integer type). This acknowledgement is then forwarded to the sending component so that it is aware that the message has been successfully sent. The underlying RPC operations that involved in message sending have been omitted for simplicity. Detailed description can be obtained from [9].

Although the centralized hub may have some communication drawbacks, for STARFISH AUV which consists of multiple computation nodes across Ethernet network, the centralized hub does provide some flexibilities for the overall system architecture. For example, under certain conditions when C2 system is required to perform heavy computational tasks, one or more of the C2Components can be loaded in different computing nodes across the AUV to help to off load the computation burden on a single processing unit. Besides that, such an implementation also allows new C2Components

to be added into the C2 system in future with minimum modification to the overall communication framework because only the C2Server needs to be notified and the new C2Component needs to be registered before it can start functioning. Of course, the supervisory components have to be modified so that it is aware of the presence of the new C2Component in the control structure.

4.5 Summary

This chapter presented the software architecture of the proposed C2 system. Component-based design principle is adopted for the construction of the control modules known as C2Components. Every C2Component defines its own data structure and performs specific algorithms or computational logics depending the assigned control task. Besides that, all the C2Components have a finite state machine which processes their tasks continuously depending on the current state of the component. The transitions between states are triggered by commands from components at higher control hierarchy and/or component's internal events. The C2Components must not have knowledge about the content of other components, and their internal state and event do not interfere with other components.

The communication among the C2Components in the C2 system is achieved via message passing protocol over an Ethernet network. All the components in the C2 system are equipped with two input and output methods for the purpose of communication. A communication object - the C2Msg is defined to encapsulate the data types that can be exchanged among the components. This provides the components with a standardized communication message and make future expansion possible. Instead of connecting the C2Components directly for communication, they are connected to

a centralized software server - the C2Server. Besides acting as the message routing hub, the C2Server also configures C2Components at startup.

Chapter 5

Simulation Studies

5.1 Introduction

Simulation is always helpful for validating developments and implementations before final deployment. In this chapter, the STARFISH simulator built by the ARL for the STARFISH project is described in Section 5.2. Several simulation cases and missions have been designed and tested on the simulator to verify the functionalities of the command and control system. During the simulation trials, the AUV is given a mission to dive at the start point; navigate though a few mission points at certain depths and finally, surface at the end point. The purpose of this mission trial is to 1) test the functionalities of the components in the C2 system and 2) verify the overall C2 system's performance in carrying out an assigned autonomous mission. The missions and corresponding results are discussed in the following sections.

5.2 The STARFISH Simulator

The C2 system designed and developed has been programmed and tested on the STARFISH simulator - a 3D System-In-The-Loop (SITL) simulator that uses Open Dynamic Engine (ODE) [1] and was built specifically for STARFISH project. SITL simulation has the advantage of "plug and play" capability when comes to system testing and deployment. The same copies of codes or programs that are running in the SITL simulator can be ported directly into the hardware platform for testing without any alteration. The concept has been tested successfully with our first AUV prototype (Fig.6.1 (b)) described in the next chapter. This results in rapid and simplified system development.

The STARFISH simulator provides a GUI for the AUV as shown in Fig. 5.1. This enables live data and simulated environmental visualization during mission execution. Different sea floor terrain can be created and simulated in the simulator for testings in different scenario. A configuration file is provided for simulator's parameter assignment while simulated data can be logged in separate file for post-simulation analysis. Besides that, the simulator also allows different underwater conditions like sea current to be created for more realistic testing.

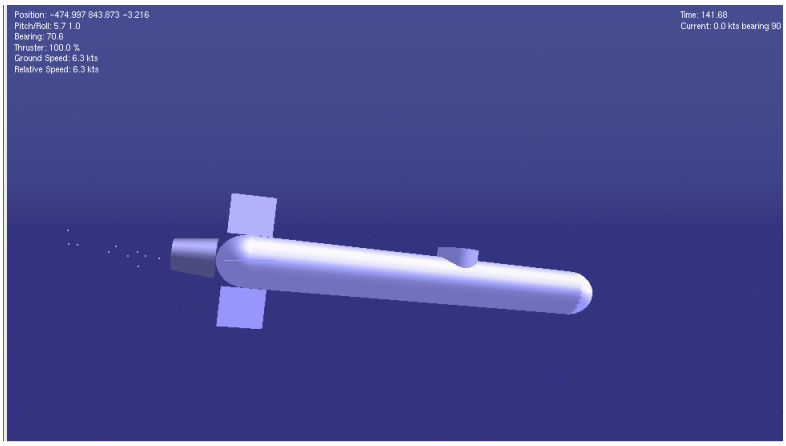


Figure 5.1: Screen capture of STARFISH simulator.

5.3 Path Following and WayPoint Following Navigation

Vehicles operating on the water surface or underwater are subjected to the disturbance from sea current or wind. In order to get to the target position, the vehicle has to utilize either path following or waypoint following navigation. Waypoint following navigation tries to adjust the vehicle's bearing periodically so that the tail-head axis of the vehicle is pointing towards the target position. On the other hand, path following navigation compensates the disturbance by adjusting the vehicle's bearing so that the resultant heading of the vehicle's movement is pointing towards the target position.

In this simulation, the AUV is given a simple navigation mission going through several mission points within the mission area. The simulation is repeated twice with the same setup and thrust setpoint throughout the mission. The first simulation is conducted without the sea current while during the second simulation, the AUV is subjected to simulated sea current of 2 knots, at a bearing of 90 degree. The

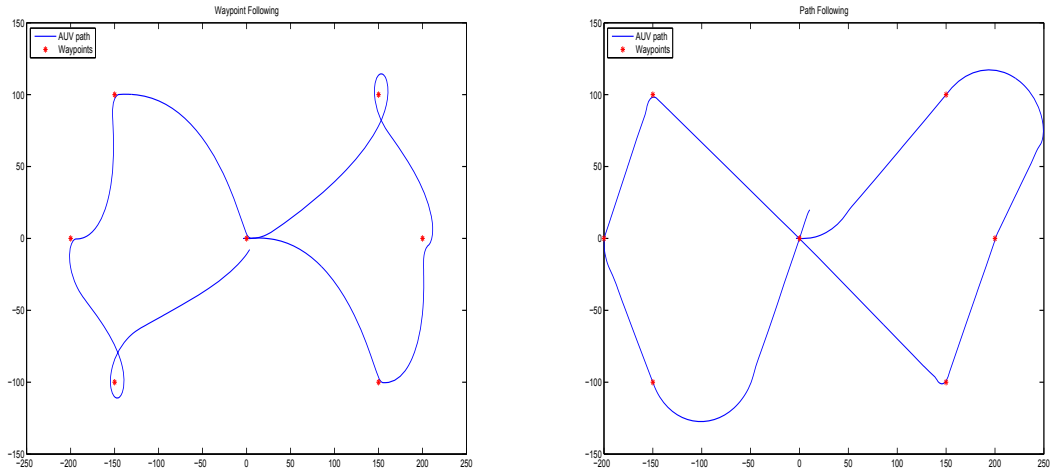
simulations validated the pilot component (Path_Executor) within the C2 system in performing different modes of navigation.

5.3.1 Simulation Results

Fig. 5.2 shows how well both the navigation modes perform in a simulated environment. From Fig. 5.2-(a), although the AUV managed to complete the mission, the navigational pattern is not as expected. The resulted AUV trajectory is not smooth throughout the mission and the AUV is forced to circulate about two of the waypoints because of the disturbance caused by the sea current. This is inefficient as the AUV had to make extra effort to point its bearing towards the following target position.

On the other hand, path following navigation mode (Fig. 5.2-(b)) has shown more promising results under the disturbance of sea current. The AUV's output trajectory is smoother and there is no circular motion around any of the assigned waypoints. Although larger turning radius is observed from the 2nd waypoint to the 3rd waypoint, and again from 7th waypoint back to the start location due to the AUV's bearing at that instance as well as the sea current, this mode of navigation is preferred over the earlier mode. The problem of larger turning radius can be avoided if subsequent waypoints are made available to the pilot so that early compensation can be made before the AUV reaches the current target waypoint.

Overall, both the navigation modes performed reasonable in the simulator. This validated the navigational functionality of the pilot.



(a) (b)

Figure 5.2: Path following and WayPoint following navigation with current = 2 knots at bearing of 90 deg. (a) waypoint following navigation. (b) path following navigation.

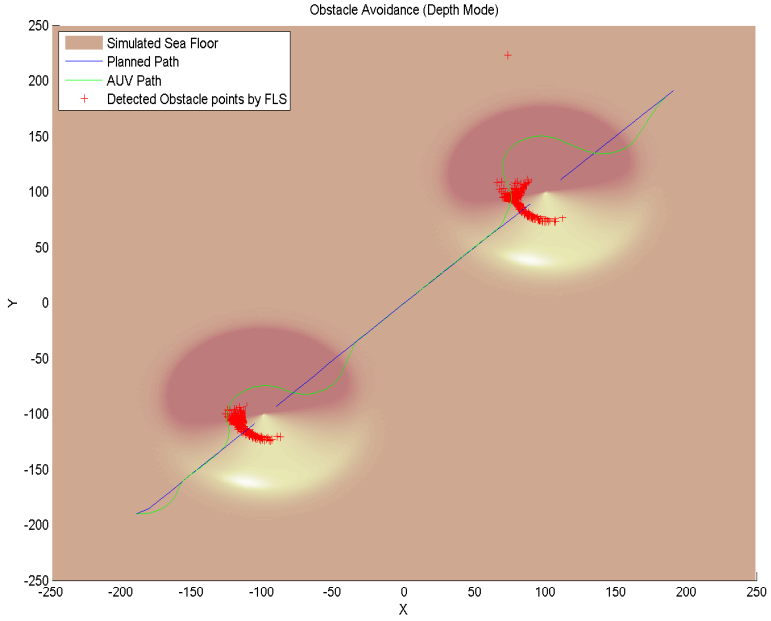
5.4 Obstacle Avoidance

In this part of simulation, obstacle avoidance behavior of the C2 system is tested. There are two modes of obstacle avoidance behaviors built in the pilot component: depth keeping mode and altitude keeping mode. Depending on the requirement of the mission, either one of the behaviors can be activated. The simulations are carried out first with path planning from the start position to the end position in a map without any obstacles. However, after a straight path is obtained, the mission starts with another sea map with the presence of two simulated obstacles protruding from the sea floor. If the planned path is passing through the obstacle, the obstacle avoidance behavior is initiated. After the AUV is clear of the obstacle, it resumes the mission and continues with its planned path.

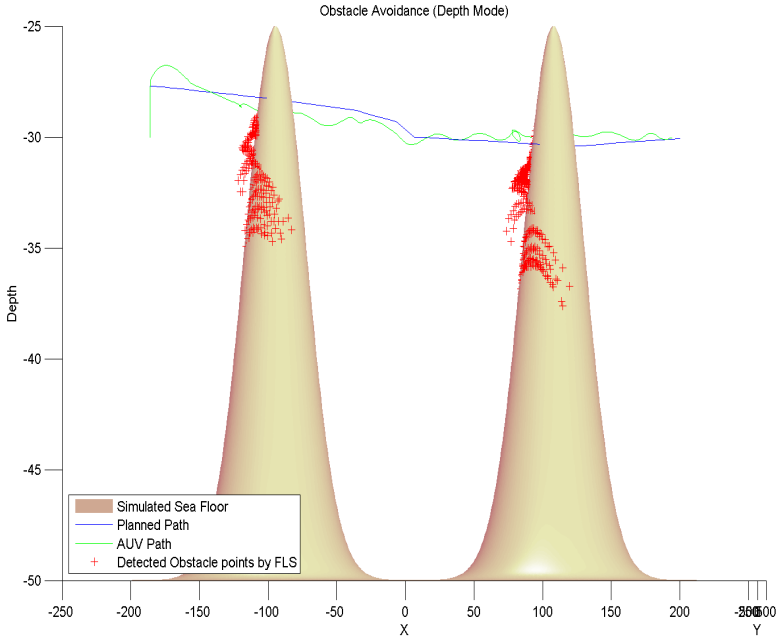
5.4.1 Simulation Results

Fig. 5.3 shows the top and side views of a simple navigational mission with depth mode obstacle avoidance behavior. The red marks are the obstacles detected by the AUV's FLS sonar, the blue line is the straight planned path between the start and end points while the green line is the output trajectory performed by the AUV. Since the AUV strived to maintain its depth, it has no choice but to avoid the obstacles by going around them. From the top view (Fig. 5.3-(a)), it is clearly seen that the AUV started to navigate away from its pre-planned path when it detected the obstacle. No collision happened throughout the whole mission. Although some deviation occurred between the resulted AUV's depth and the path's depth as seen in Fig. 5.3-(b), they are acceptable considering the obstacle size and the AUV dynamics.

In the altitude control obstacle avoidance mode, the AUV tried to keep its altitude with respect to the sea floor, and in this case, relative to the height of the obstacle. Thus, instead of going around the obstacle, the AUV tried to go over the obstacle. Fig. 5.4-(a) shows the top view of the resulted output trajectory from the mission. Since both the missions for depth and altitude control obstacle avoidance modes start and end at the same position, their pre-planned path is the same. It can be seen that the AUV stayed on the pre-planned path throughout the mission even when it is avoiding the obstacles. From Fig. 5.4-(b), the obstacle avoiding maneuver is evident when the AUV is close to the obstacle. Again, the AUV did not collide with any of the obstacles during the mission. Whenever the AUV has passed the obstacle (threat is cleared), it continued with the rest of the pre-planned mission points' depth.

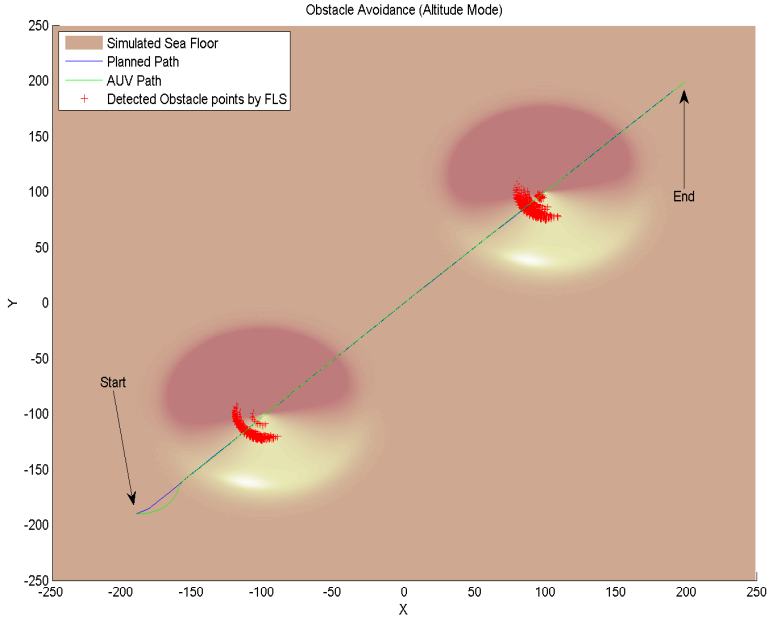


(a)

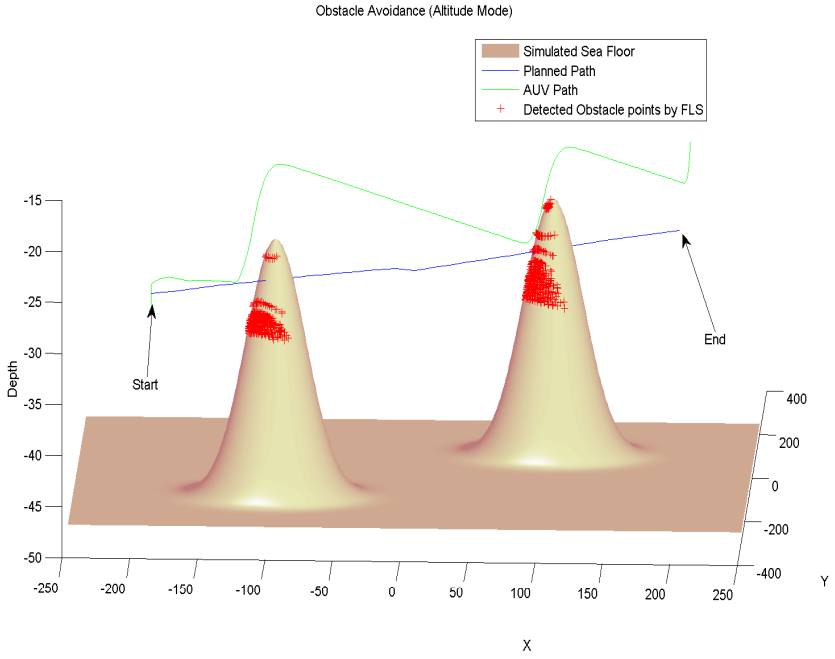


(b)

Figure 5.3: (a) Top-view of obstacle avoidance in depth control mode. (b) Side-view of obstacle avoidance in depth control mode.



(a)



(b)

Figure 5.4: (a) Top-view of obstacle avoidance in altitude control mode. (b) Side-view of obstacle avoidance in altitude control mode.

5.5 Station Keeping

Station keeping or position maintaining around an point is useful when the AUV is waiting for operator's command or in situations when the AUV needs to perform intensive computation like decision making or replanning. The AUV in STARFISH project is a single thruster, torpedo-shape underwater vehicle and is not designed for accurate and stable station keeping behaviors, having certain degree of position maintaining within an acceptable vicinity of a given point is still desirable. This not only allows the operator to send command for mission change or replanning while the AUV is operating underwater, but also ensuring that the AUV would not be drifted by the sea current.

In this simulation, two modes of station keeping were tested. First is station keeping without the sea current while second is with the sea current. Since the STARFISH AUV has a slightly positive buoyancy, turning off the thruster for station keeping even there is no sea current will not leave the AUV to stay at the same position. Thus, for station keeping without the sea current, the thruster is turned on and the AUV has to move in a circular maneuver so that enough downward torque can be generated by the fins to keep the AUV at the specified depth and around the station keeping location.

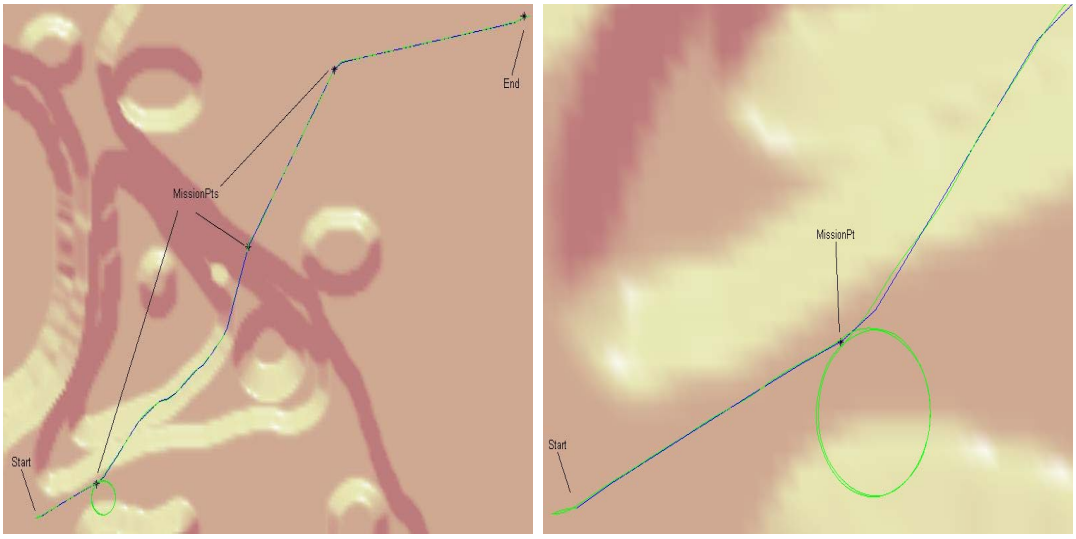
However, with sea current, the AUV may be carried away by the current if it is moving in a circle to keep station. Instead of doing so, the AUV is instructed to thrust forward against the direction of the sea current when it is behind the station keeping position and stop the thruster when it is beyond the position so that the sea current will push it backward again. This forward-and-backward motion of the AUV around the position of station keeping makes sure that the AUV is within the point's

vicinity.

5.5.1 Simulation Results

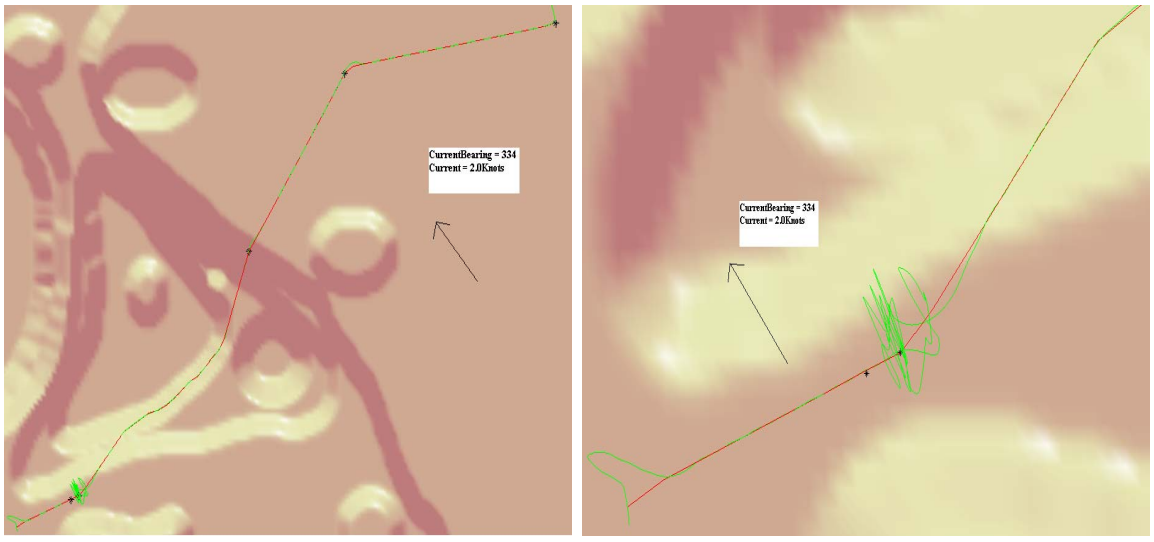
The two simulations are conducted in STARFISH simulator and their output trajectories are plotted and shown in Fig. 5.5 and Fig. 5.6. Note that for the simulation with sea current, the current speed is assumed constant at a fixed angle throughout the test. For both the missions, the AUV is instructed to perform station keeping when it reaches the first mission point. Fig. 5.5-(a) shows the overall trajectory output for the mission while Fig. 5.5-(b) is the close-up around the first mission point where the station keeping is performed. The AUV is able to navigate in a circular motion once it gets to the mission point. When the time for station keeping is up or when the AUV receives command from the operator to resume the mission, it continues with its planned path to the next mission point. The AUV behaved as expected in station keeping without current.

with a sea current of 2 knots at an angle of 334 degree, the AUV behaved differently in station keeping (Fig. 5.6). When the AUV reached the mission point for station keeping, it first adjusted its bearing so that its heading into the opposite direction of the sea current. Once the AUV is in front of the point of station keeping, the thruster is turned off to let the current to push the AUV behind the station keeping point. The process is repeated as long as the AUV is in station keeping mode and, continued the mission when command received from the operator or the duration for station keeping has elapsed. Although the result does not show station keeping over a straight line along the point of station keeping, the area occupied during the operation is acceptable.



(a) (b)

Figure 5.5: (a) Station keeping without current. (b) Zoom-in of the station keeping without current.



(a) (b)

Figure 5.6: (a) Station keeping when current exist. Current speed is 2 knots, current bearing is 334 deg. (b) Zoom-in of station keeping when current exist.

5.6 Mission Abort

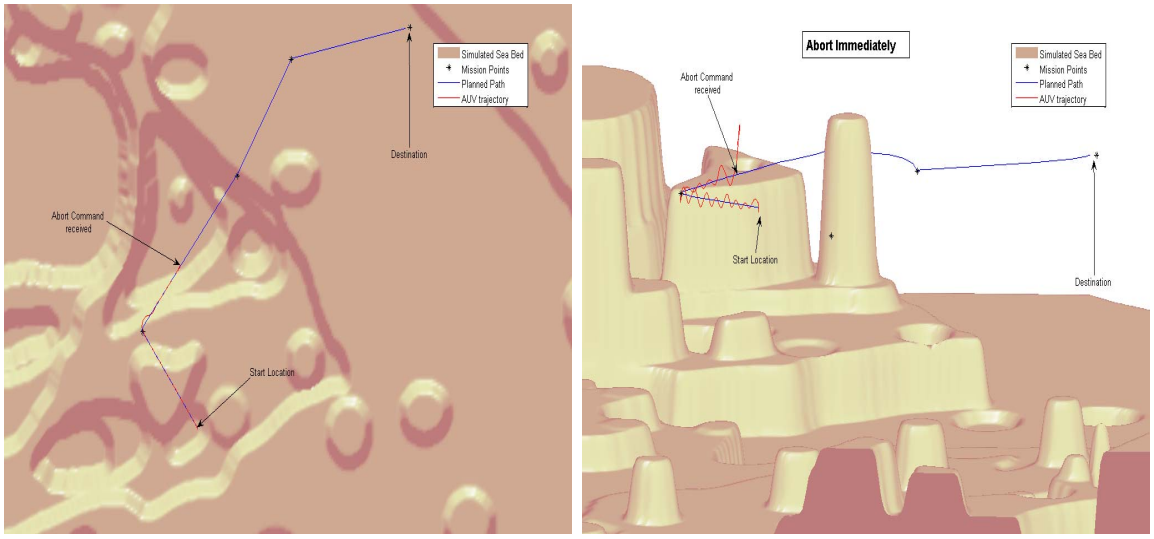
During mission execution, there are situations when the AUV may encounter problems that it can not handle. Under such circumstances, it is safer for the AUV to abort the mission and surface. Besides that, the operator can also send abort commands to the AUV if necessary during the time when it is in the water. Currently, the C2 system provides 4 different types of abort commands for the operator. In emergency, the operator can instruct the AUV to abort immediately and surface at its current location. However, the AUV can also be instructed to abort the mission and surface/navigate to the next mission point, the start location or the current mission's destination.

All of the mission abort types are tested in the simulator for a mission with a total of four mission points. The abort commands are sent by the operator through CCL message during the mission execution. Once the abort command is received, the AUV discarded the current mission path and surfaced at the assigned location.

5.6.1 Simulation Results

Fig. 5.7 shows the top and side views of the AUV's output trajectory when it was instructed to abort immediately. Whenever an abort mission command is received, the AUV discards its current mission path and surfaces. In the case of immediate mission abort, the AUV just surfaces. It is then the operator's task to retrieve the AUV based on the last location updates or through the AUV's GPS buzzer.

The AUV's output trajectory to abort at next mission point of the current mission is shown in Fig. 5.8. The abort command is received when the AUV is navigating from its start location to the first mission point (which is also its immediate next

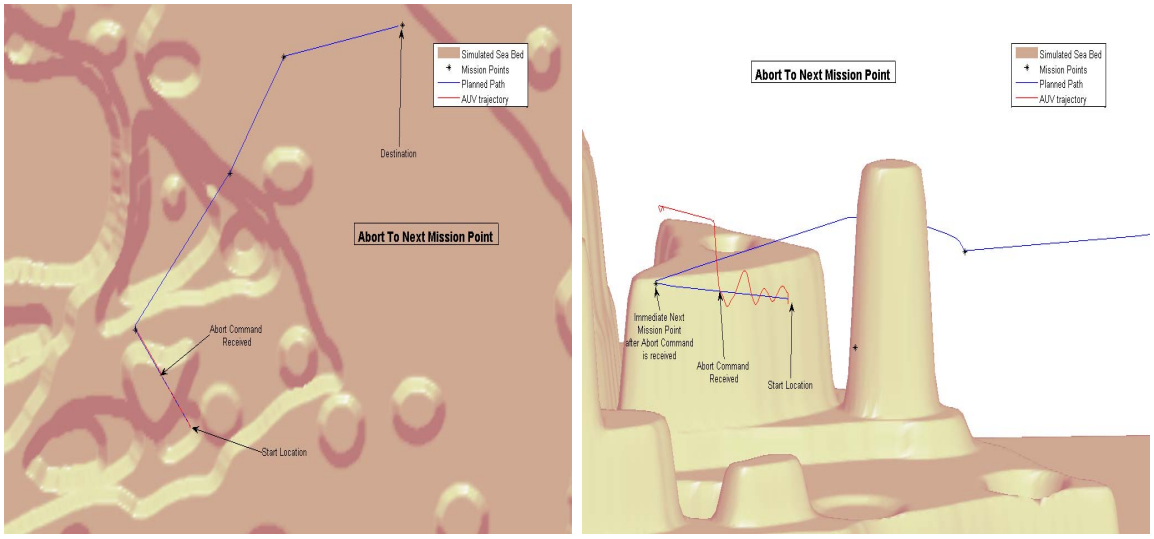


(a) (b)

Figure 5.7: Abort mission immediately. (a) Top-view of the mission path and AUV output trajectory. (b) Side-view of the mission path and AUV output trajectory.

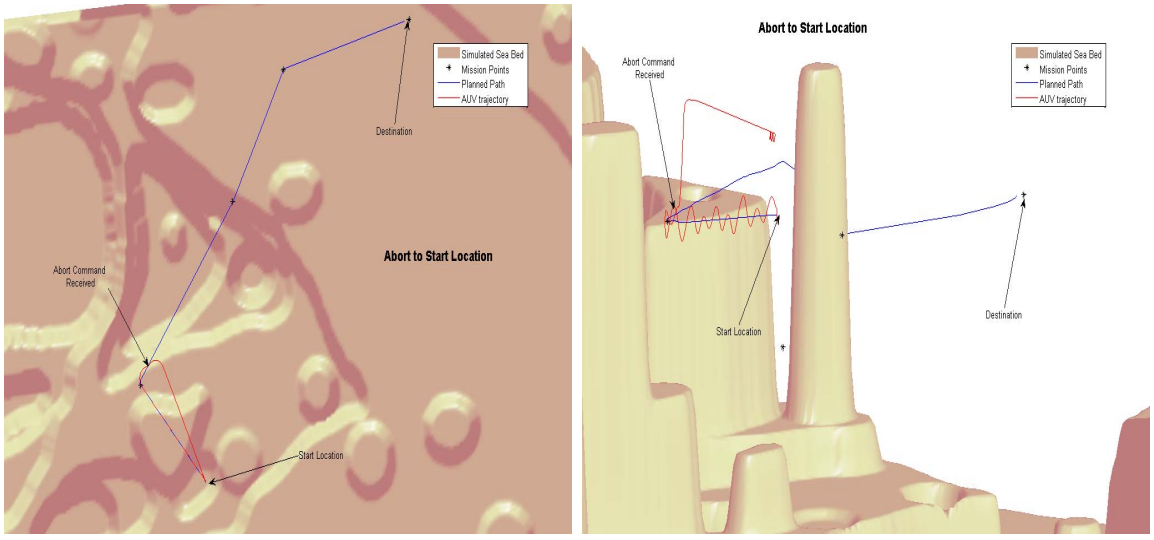
mission point for this case). As can be seen, the AUV surfaces and continues to navigate to the first mission point before it stops.

Fig. 5.9 and Fig. 5.10 show the results of the AUV aborting to its start location and destination. The start location is the point when the AUV received mission start command. It is also the first position taken for path planning to the first mission point. Destination is the location of the current mission's last point. Both points are recorded when mission is started. Again, the AUV behaved as expected to surface and navigate to its assigned location before stopping the thruster. Both of the mission abort types are useful to the operator so that the final location of the AUV, after a mission is aborted, is known.



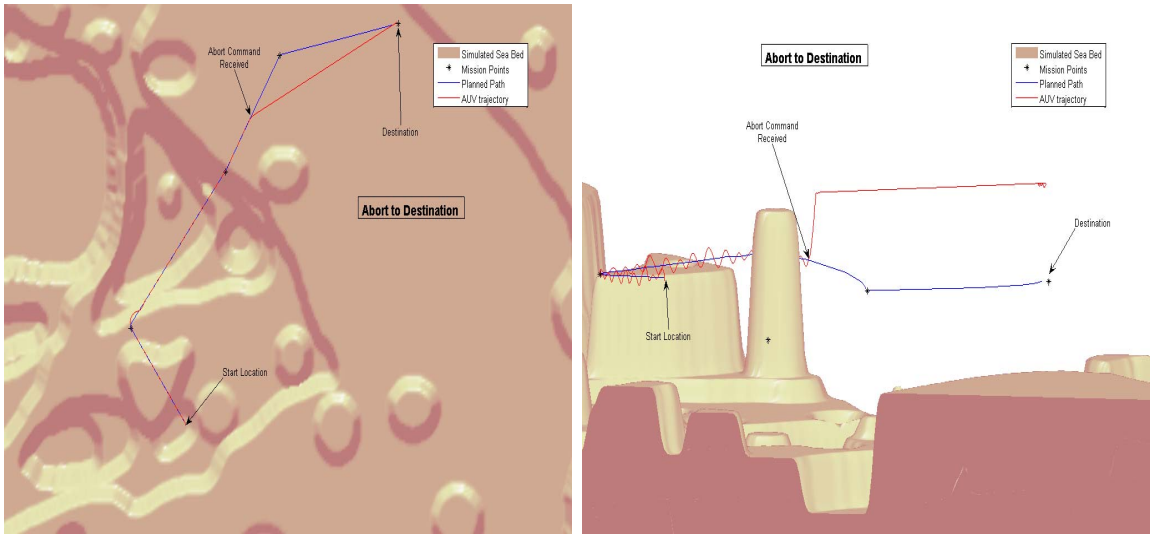
(a) (b)

Figure 5.8: Abort mission and surface/navigate at the immediate next mission point. (a) Top-view of the mission path and AUV output trajectory. (b) Side-view of the mission path and AUV output trajectory.



(a) (b)

Figure 5.9: Abort mission and surface/navigate to the AUV's mission start location. (a) Top-view of the mission path and AUV output trajectory. (b) Side-view of the mission path and AUV output trajectory.



(a) (b)

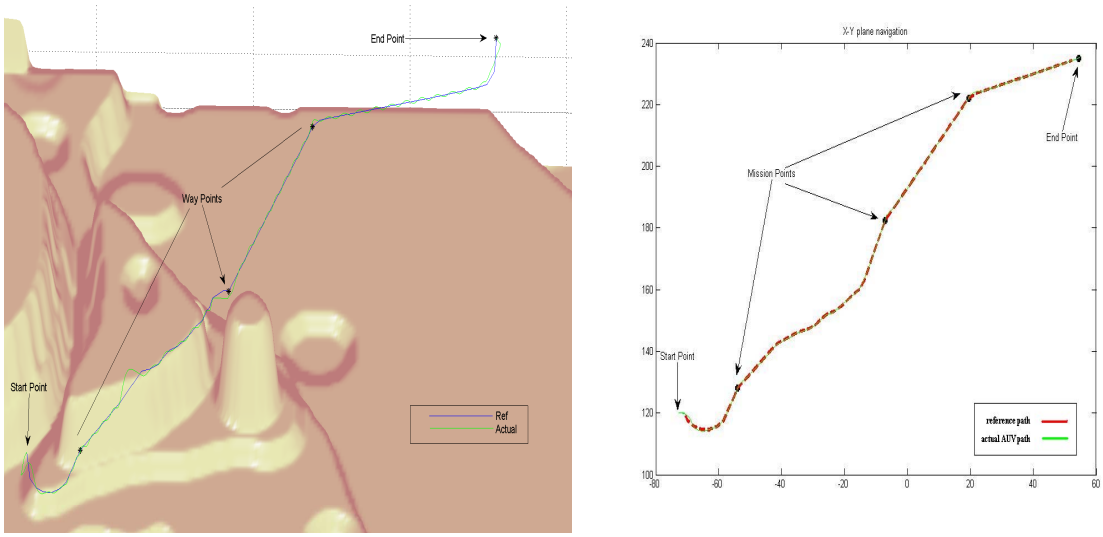
Figure 5.10: Abort mission and surface/navigate to the current mission's destination location (last point of the mission path). (a) Top-view of the mission path and AUV output trajectory. (b) Side-view of the mission path and AUV output trajectory.

5.7 Full Mission

A full mission has been carried out using simulated ocean terrain extracted from one part of the Singapore sea map [16]. In this section, an example of the execution of a complete mission with the C2 system is presented. Detailed operation and coordination among the C3Components is explained as follows:

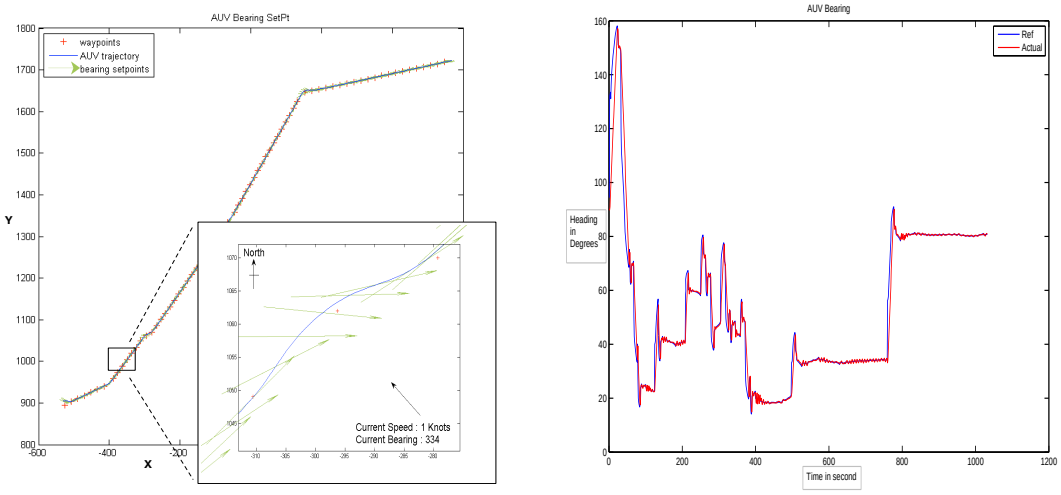
When the mission starts, the Signaling_Officer first receives start command and mission number in the form of CCL [12] message from the mother ship. It then decodes the message into C2Msg object and relays to the Captain. After making sure that all other components have been started and in standby state, the Captain sends a command to the Task_Planner for task decomposition and mission path planning

according to the mission script. Once a valid path is found, waypoints along the mission path is generated and passed on to the Path_Executor for navigation. Based on the waypoints given and sensor information, the Path_Executor generates appropriate commands to navigate the AUV along the mission path and to maintain the desired depth. Throughout the mission execution, there are few tasks being performed periodically. The Signaling_Officer sends AUV status and segments of mission path to the mother ship so that the operator is aware of the AUV's current condition as well as the mission progress. The Obstacle_Detector reads the data from FLS and check for the presence of obstacles in the mission path. The Safety_Officer pulls data from the Monitor_Server to check on AUV's health status and informs the Captain for any abnormality found (The health condition of all the devices are simulated in the simulator). The data acquired during the trials were collected and logged for post mission analysis.



(a) (b)

Figure 5.11: (a) Three Dimensional (3D) view of the mission reference and actual AUV path. (b) Top-view of the reference and actual AUV path.



(a) (b)

Figure 5.12: (a) Plot of AUV's trajectory and bearing setpoints. (b) Plot of reference and actual vehicle bearing during mission.

5.7.1 Simulation Results

Figure 5.11 shows the simulated ocean terrain, the resulting mission path and trajectory executed during the mission. Figures 5.12-(a) and 5.12-(b) show the resulted AUV's trajectory and bearing of the AUV throughout the mission execution. The AUV traveled a distance of approximately 1000 meters, with depth ranging from 0 to 15 meters under the sea surface and average bearing error under 5 degrees. As can be seen in the zoomed in area of Fig. 5.12-(a), the bearing setpoints are made to compensate the sea current so that the vehicle exhibits path following behavior. The resulting mission path took into account of vehicle's maximum turning and pitching angle. All the waypoints were visited when the mission was completed. Although these are preliminary results obtained with a simple mission, it is sufficient to verify the basic functionality of each component while confirming the

integrity of the overall C2 system.

5.8 Summary

This chapter described the STARFISH simulator developed for system testing and validation as well as the simulation results for several simulated missions designed to validate the overall functionalities of the proposed C2 system. Initially, simulations are carried out to evaluate the AUV's capabilities in performing important maneuver behaviors like path following with and without the presence of sea currents, obstacle avoidance and station keeping. Later, different types of mission abort commands are tested to ensure the AUV's safety when required. Finally, A complete navigational mission was carried out in a simulated ocean terrain for the overall C2 system evaluation.

The results from the simulations shown correct behaviors for the C2 system. However, further refinements are required to improve the performance of the control system. Although the complete navigational mission is simple, it is sufficient to fulfill the requirement of mission one mentioned in Section 1.2.

Chapter 6

Field Trials

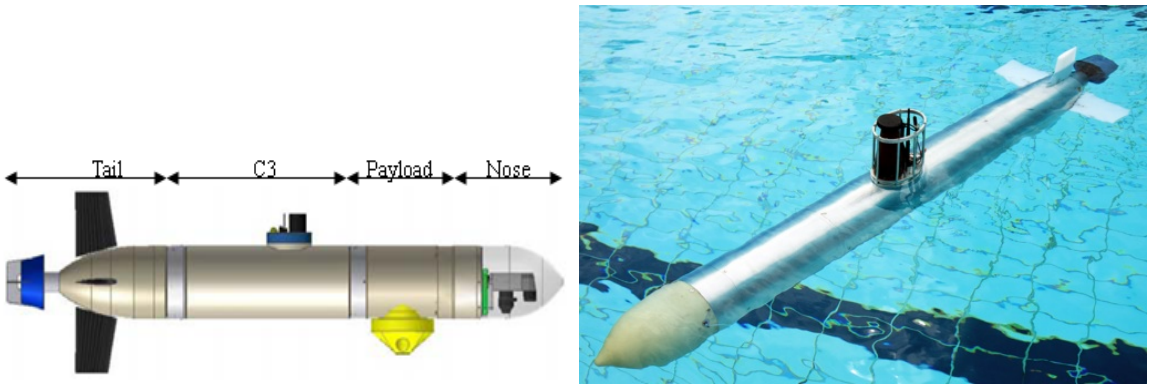
6.1 Introduction

The current stage of the STARFISH project has yielded the first AUV prototype. This enables the command and control system to be deployed into the real hardware and tested in field trials to further validate its functionalities. Section 6.2 provides a brief description of the AUV's hardware implementation. This is followed by several field trials that were conducted at the Pandan Resevoir, Singapore. The results from the trials were analyzed and discussed in detail in the following sections.

6.2 Hardware Implementation: The STARFISH AUV

6.2.1 Mechanical Design

The STARFISH AUV [19] has the shape of a torpedo and it is made of different sections, interconnected by means of a custom designed interlocking teeth arrangement and made water tight by using a double piston o-ring seal (Fig. 6.1). The hull is made of Aluminium Alloy while the nose cone is made of Delrin. It is 0.2m in diameter and the length is restricted from 1.45 to 2m depending on the payload sections. The



(a) (b)

Figure 6.1: (a) STARFISH AUV System Configuration. (b) STARFISH AUV at pool test.

fineness ratio of the AUV hull varies from 7 to 10. The AUV weighs approximately 34.5kg in air and offers approx. 40kg of buoyancy. A custom designed emergency ballast capable of holding trimming weights is built on the nose section to adjust the buoyancy as required and released in time of emergency so that the AUV has enough buoyancy to return to the surface.

The AUV's nose section comprises of dry and wet subsections. The wet subsection is free flooded and houses sensors like altimeter, Foward Looking Sonar(FLS) and pressure gauge while the dry subsection acts as the connection and communication link between the C3 (Command Control and Communication) section and nose section. C3 section contains the command, control and communication PC along with related sensors. Besides, it also houses batteries that provide power for the operation of entire AUV. A communication tower that consists of antennas for GPS, wireless LAN, GSM modem and an Acoustic Modem transducer is built on top of the C3 section to provide communication with the mothership/operator. The AUV is propelled by a single DC motor and is mounted on the tail section together with the fins which

are controlled by individually using servo motors. This section is partially flooded to provide water cooling for the main thruster. These three main sections are interlinked by the Ethernet backbone, Power bus, and the Run level bus.

6.2.2 Computer and Electronic module

The building of STARFISH AUV adopted modular principles from mechanical, electrical and software perspectives. There are two types of modules: hardware modules that implement software drivers to communicate with the electronics and software modules that execute algorithms for AUV control. Every section consists of multiple modules that are connected locally by the Ethernet switch. Modules from different sections can communicate via the Ethernet backbone. There is a Pentium class PC 104+ with self complied embedded Linux residing in the C3 section and is responsible for running core softwares. Besides that, other sections have Microcontroller Units (MCU) that act as the hardware interface with the sensors and actuators.

6.2.3 Power System and Sensor Suite

The STARFISH AUV carries a total battery capacity of 1.35kWh with Lithium Polymer (Li-Po) as primary energy source. The amount of power is sufficient for autonomous mission of approximately 2 hours. Different sections obtain power from the power bus which runs all along the length of the AUV and provides 48V of power. Efficient DC to DC converters are used to convert the power as per the requirement of different sections.

Sensors play an important role in AUV to make sure that it achieves mission objectives as well as keeping the AUV safe throughout the mission. The STARFISH AUV is equipped with a complete sensor suite. Firstly, the nose section is mounted

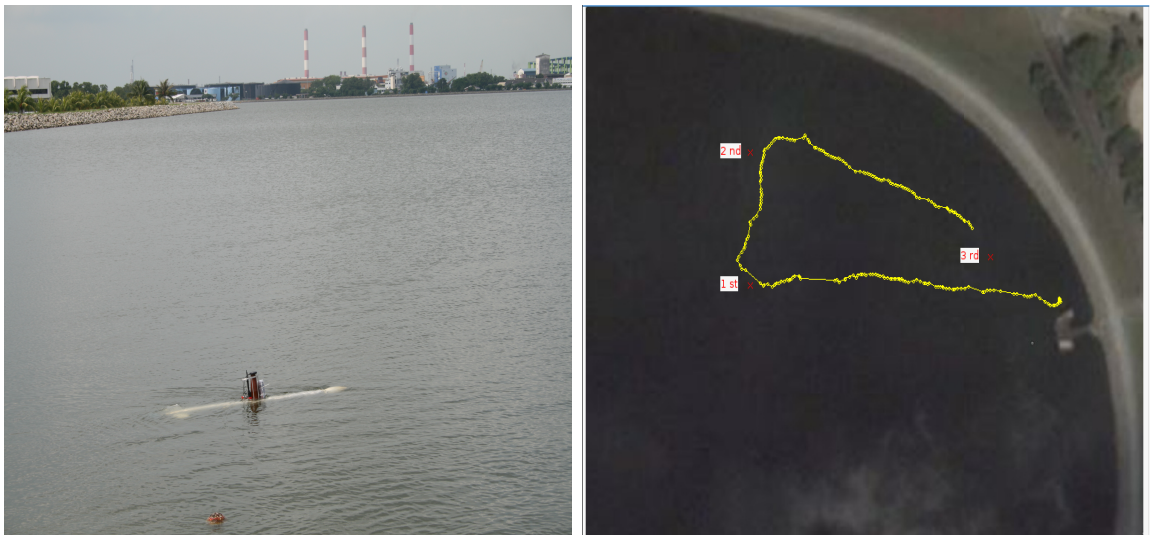
with depth sensor, altitude sensor, pressure sensor and obstacle avoidance sonar. The C3 section comes with a compass and an Inertial Measurement Unit (IMU) for navigational purposes. In order to monitor the AUV's condition during mission, all the sections are equipped with leak detector and temperature sensor to make sure no leakage and the electronics are functioning below the safe temperature limits. Since the STARFISH AUV can be extended with different payload sections, other sensors like Doppler Velocity Log (DVL) and side scan sonar can be attached to provide more functionalities. In the AUV prototype, the DVL is attached as payload section to provide better positioning capability.

However, the AUV requires only three basic sections which are Tail, Nose and C3 sections to function. Different payload sections can be added while the Tail or Nose sections can be exchanged later depending on the requirement of the mission, provided that the final setting has the three basic sections. This design has the advantages of extensibility, modularity as well as flexibility because it allows different sections to be included or exchanged with minimum effect to the software behavior as well as the overall AUV system architecture.

6.3 Field Trial

A simplified surface field trial has been carried out to validate certain functionalities of the C2 system. The trials were conducted at Pandan Reservoir ($Latitude = 1.3171°$, $Longitude = 103.7482°$), Singapore (Fig. 6.2) using the first STARFISH prototype. In the trial, the AUV is given a mission file to navigate through three mission points and is expected to stop once the mission is completed. Since it is a surface mission (depth = 0), the AUV is subjected to wind disturbance as well as north-pushing

current caused by a whirlpool near the start location. The expected behavior is for the deliberative layer to receive operator's command and plan a path through the mission points, while for the reactive layer to maneuver the vehicle based on the mission path and abort the mission if any abnormity happens.

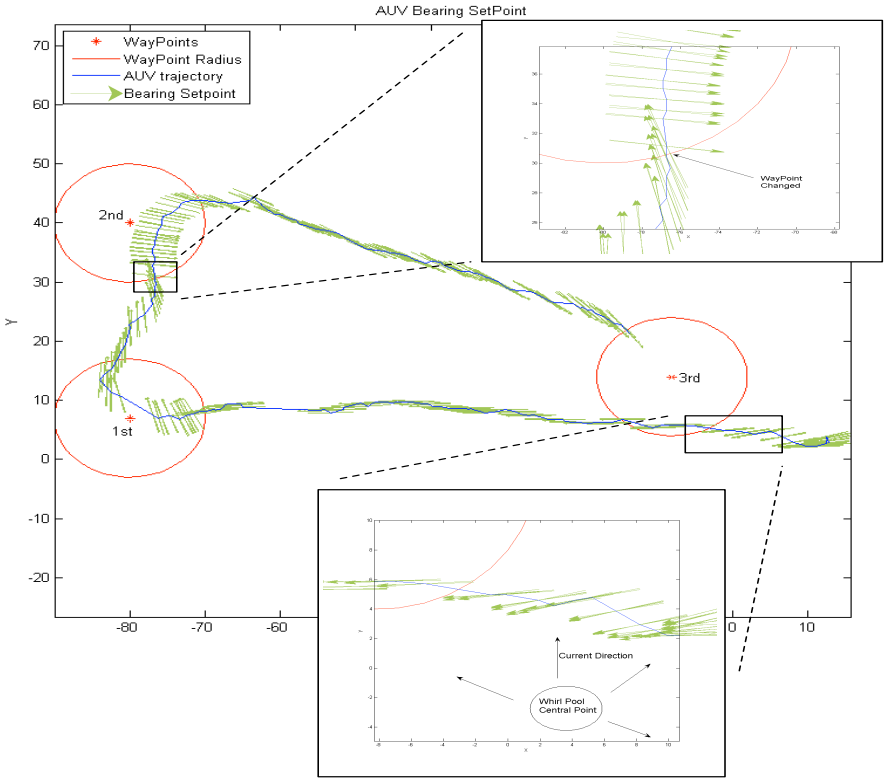


(a) (b)

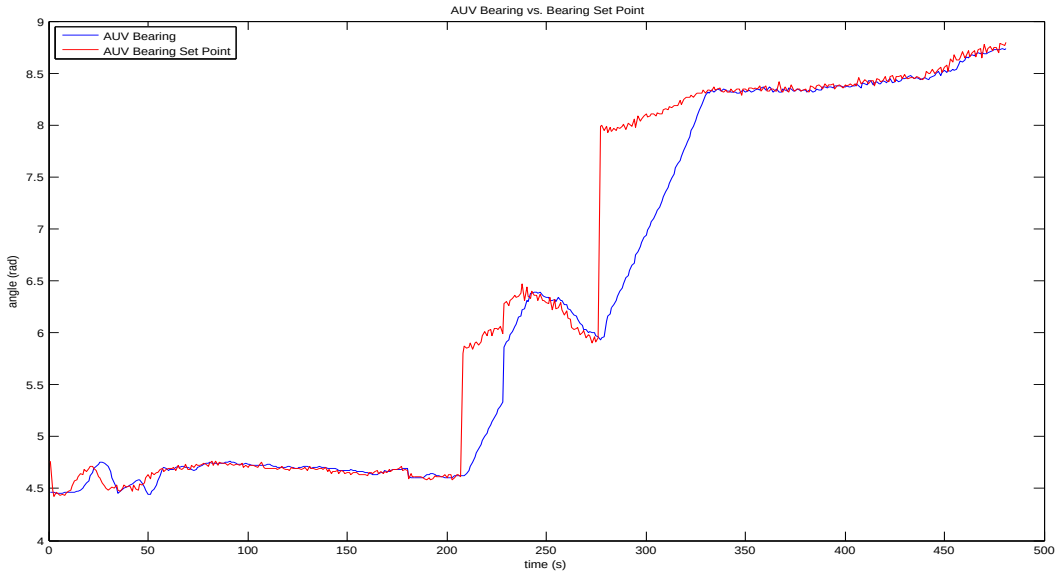
Figure 6.2: (a) Photograph of STARFISH AUV during field trial. (b) Plot of AUV's mission path in a lake test. The coordinates are based on raw GPS data received by the AUV during the execution. The AUV started from the floating platform and navigated through all three mission points.

Fig. 6.2-(b) shows the resulted trajectory (yellow dots) executed by the AUV. The AUV's positioning is based on raw GPS data while the velocity and heading is obtained from the DVL. Although the GPS lost its fix for a few times during navigation (straight yellow line), the AUV managed to complete the mission.

The resulted AUV's trajectory is plotted in Fig. 6.3-(a). Green arrows and the blue line show the AUV's bearing set points and the path executed. Red circles are the waypoint radius. The waypoint is considered reached when the AUV is within this



(a)



(b)

Figure 6.3: (a) Plot showing the AUV's bearing setpoints (green arrows) during navigation. Red circles are the waypoint radius, the waypoint is considered reached when the AUV is within the circle. (b) Plot shows the AUV's bearing and bearing setpoints through the mission execution.

area. The AUV's bearing is set slightly towards the south-west direction at the start location to compensate the whirlpool current. Once it is out of the whirlpool area, the set points pointed directly towards the first mission point. This demonstrated the path following behavior implemented in the reactive layer. Also, it can be seen that the bearing setpoints changed to point to the next waypoint whenever the AUV is within the waypoint radius.

The AUV's commanded change in bearing is showed in Fig. 6.3-(b). Although there is a delay in the response compared with Fig. 5.12-(b), it is expected in hardware implementation. Despite the simplified surface navigational mission, the C2 system has shown correct behavior so far. We expect to further validate the overall C2 system's functionalities in our future trials.

6.4 Summary

In this chapter, the STARFISH first prototype AUV is described. The proposed C2 system has been deployed in the AUV and a field trial has been conducted at Pandan reservoir, Singapore. Since the prototype AUV is still undergoing hardware and software fine-tuning, only simple surface missions can be carried out to test the functionalities of the developed C2 system. The prototype AUV has successfully completed a simple surface run navigating through three mission points within the lake based on raw GPS data. More field trials will be carried out in the near future when all the components in the AUV is working properly.

Chapter 7

Conclusions and Future Work

A novel command and control system architecture has been developed for AUVs in the STARFISH project. The focus at current stage of the project has been to develop a generic control and software architecture for a single AUV and later, expended to multi-AUV operations.

The design, testing and validation of the architecture has been described. The proposed control architecture has a hybrid structure. Control modules are grouped into three hierarchical control layers that enables the mission supervision and command to be executed at a higher layer while decouples the vehicle and navigational control from the lower layer. It also provides capabilities for real time mission status updates and vehicle or mission error detection. The control architecture also allows multiple Cheif_Scientist components to be added when needed without affecting the overall structure.

The use of Component-object based design principle for the development of software architecture provides flexibilities in terms of software implementation or alternation. Instead of modifying the existing software components, new components

with identical interfaces but different algorithms can be built and loaded when necessary. Inter-component communication using RPC protocol via Ethernet network allows components with high computational requirements to be loaded in different processing nodes across the network.

The proposed C2 system has been developed and tested in a software simulator. Since the STARFISH project is still at its early stage, the test is only limited to a single AUV performing simple navigational tasks. A field trial has been conducted at Pandan reservoir, Singapore to further validate the functionalities of the developed C2 system a prototype AUV. Although the field trial only involved a simple surface navigational mission, it is able to show the expected maneuver behavior and identify areas for further enhancement in the future.

Future work consist of refining the current C2 system and implement the Cheif_Scientist and the Scientist components. We also expect to proceed to sea trial once the results from the field trials are satisfactory. When the second STARFISH prototype is available, the current version of C2 system will be enhanced and expanded so that it is capable of handling multi-AUV missions.

Bibliography

[1] *Object dynamic engine*, available at: http://www.ode.org/. Accessed 1 August 2008.

[2] A. Alvarez, A. Caiti, and R. Onken, *Evolutionary path planning for autonomous underwater vehicles in a variable ocean*, Oceanic Engineering, IEEE Journal of **29** (2004), no. 2, 418–429.

[3] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and Woo-Keun Yoon., *Rt-component object model in rt-middleware; distributed component middleware for rt (robot technology).*, Computational Intelligence in Robotics and Automation, 2005. CIRA 2005. Proceedings. 2005 IEEE International Symposium on. (June 2005), 457–462.

[4] R.C. Arkin, *Behaviour-based robotics*, MIT Press, Cambridge, MA., 1998.

[5] S. Bhattacharyya, R. Kumar, S. Tangirala, M. O'Connor, and L.E. Holloway, *Animation/simulation of missions for autonomous underwater vehicles with hybrid-model based*, American Control Conference, 2006 (2006), 6 pp.–.

[6] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Oreback, *Towards component-based robotics*, Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on (2005), 163–168.

[7] Davide Brugali, Alex Brooks, Anthony Cowley, Carle CÃtÃc, Antonio DomÃnguez-Brito, Dominic LÃctourneau, Francis Michaud, and Christian Schlegel, *Trends in component-based robotics*, pp. 135–142, Springer, 2007.

[8] H. Bruyninckx, *Open robot control software: the orocos project*, Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on **3** (2001), 2523–2528 vol.3.

[9] Mandar Chitre., *Dsaav - a distributed software architecture for autonomous vehicles.*, 2008.

[10] P.S. Dias, R.M.F. Gomes, and J. Pinto, *Mission planning and specification in the neptus framework*, Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on (2006), 3220–3225.

[11] Antonio DomÃ?nguez-Brito, Daniel HernÃ¡ndez-Sosa, JosÃ© Isern-GonzÃ¡lez, and Jorge Cabrera-GÃ¡mez, *Coolbot: A component model and software infrastructure for robotics*, pp. 143–168, Springer, 2007.

[12] Eberbach E., Duarte Ch., Buzzell Ch., and Martel G., *A portable language for control of multiple autonomous vehicles and distributed problem solving*, Proc. of the 2nd Intern. Conf. on Computational Intelligence, Robotics and Autonomous Systems CIRAS'03, Singapore (2003).

[13] Jonathan Evans, Pedro PatrÃn, Ben Smith, and David Lane, *Design and evaluation of a reactive and deliberative collision avoidance and escape architecture for autonomous robots*, Autonomous Robots **24** (2008), no. 3, 247–266.

[14] Xiaoming Li, Yuzhen Jin, and Xudong Hu, *An xml-driven component-based software framework for mobile robotic applications*, Mechatronic and Embedded Systems and Applications, Proceedings of the 2nd IEEE/ASME International Conference on (2006), 1–6.

[15] M. Makatchev and S.K. Tso, *Human-robot interface using agents communicating in an xml-based markup language*, Robot and Human Interactive Communication, 2000. RO-MAN 2000. Proceedings. 9th IEEE International Workshop on (2000), 270–275.

[16] Maritime and Port Authority of Singapore (MPA), *Charts for small craft . singapore strait and adjacent waterways.*, 2004.

[17] P. Ridao M.Carreras, N. Palomeras and D. Ribas, *Design of a mission control system for an auv*, International Journal of Control. **80** (2007), no. 7, 993–1007.

[18] A.M. Meystel and J.S. Albus, *Intelligent systems: Architecture, design, and control*, ch. 1-10, New York: John Wiley&Sons, 2002.

[19] M.N.Sangekar., Mandar Chitre, and T.B. Koay, *Hardware architecture for a modular autonomous underwater vehicle starfish.*, OCEANS 2008. MTS/IEEE Quebee. (2008).

[20] P. M. Newman, *Moos - a mission oriented operating suite.*, Tech. Report OE2003-07, MIT, Department of Ocean Engineering., 2003.

[21] C. Petres, Y. Pailhas, Y. Petillot, and D. Lane, *Underwater path planing using fast marching algorithms*, Oceans 2005 - Europe **2** (2005), 814–819 Vol. 2.

[22] Liu Jing. Prahlad Vadakkepat, *Particle based path planning algorithm for autonomous underwater vehicle navigation.*, submitted.

[23] P. Ridao, J. Yuh, J. Batlle, and K. Sugihara, *On auv control architecture*, Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on **2** (2000), 855–860 vol.2.

[24] O.J. Røseth, *Object-oriented software system for auv control*, Proc. of Mobile Robots for Subsea Environment; International Advanced Robotics Program. Monterey, CA. (1991).

[25] C. Vasudevan and K. Ganesan, *Case-based path planning for autonomous underwater vehicles*, Autonomous Robots **3** (1996), no. 2, 79–89.

[26] A. Yavnai, *Architecture for an autonomous reconfigurable intelligent control system (arics)*, Autonomous Underwater Vehicle Technology, 1996. AUV '96., Proceedings of the 1996 Symposium on (1996), 238–245.

[27] H. Yavuz and A. Bradshaw, *A new conceptual approach to the design of hybrid control architecture for autonomous mobile robots*, Journal of Intelligent and Robotic Systems **34** (2002), no. 1, 1–26.

[28] George A. Pizza. Yu T. Morton, Douglas A. Troy, *A state-based modelling approach to develop component-based control software for flexible manufacturing system.*, International Journal of Computer Integrated Manufacturing **16** (2003), 292–306.

[29] QiaoRong Zhang, *A hierarchical global path planning approach for auv based on genetic algorithm*, Mechatronics and Automation, Proceedings of the 2006 IEEE International Conference on (2006), 1745–1750.