DESIGN OF A ROBUST AUTONOMOUS SURFACE CRAFT FOR DEPLOYMENT IN HARSH OCEAN ENVIRONMENT

ZHI LI

Design of a Robust Autonomous Surface Craft for Deployment in Harsh Ocean Environment

by

©Zhi Li

A Thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of

Master of Engineering

Faculty of Engineering and Applied Science

Memorial University of Newfoundland

March, 2013

ST. JOHN'S NEWFOUNDLAND

Contents

Abstract

The Autonomous Surface Craft (ASC) features fast development in the past few years; however, among publications about ASCs, few discussions are about ASC robustness and especially the reliable operation of the ASC in the harsh ocean environment. Therefore, in this thesis project, a robust ASC that is mainly used for reliable operation in the harsh ocean environment offshore Newfoundland is designed. As the first ASC prototype developed in the Autonomous Ocean Systems Laboratory (AOSL), the main concentration is on reliable ASC electrical and communication system design and the ASC system testing and modelling.

The ASC on-board communication and control system implements the Controller Area Network (CAN) protocol. External communication with the dock-side computer is built on 900 MHz wireless modems. Four CAN modules are developed to work on the on-board communication network, and many off-the-shelf electrical components were chosen to build the electrical system, which include the Global Positioning System (GPS), Attitude and Heading Reference System (AHRS), Weather Station (WS) and the mbed^{TM} microcontroller. Time synchronization of separate CAN modules inside this CAN network is addressed using the presented time reference message (TRM) based synchronization mechanism, and the achieved characteristics are validated using a DPO4034 oscilloscope. The wireless communication link plays an important role in ASC testing, and it can be used to transmit the supervisory command and ASC sensor

data between the ASC and the dock-side computer. To support this feature, a Matlab based Graphic User Interface (GUI) is designed to work on the dock computer as the control terminal and the display monitor of the ASC status data. A hand controller is integrated into this GUI for intuitive control of the vehicle, and the ASC position can be shown in quasi-real-time in Google Earth software.

A hydrodynamic 3 Degrees of Freedom (DOF) nonlinear model for describing the motion of the ASC is generated. Two methods, including the Taylor series expansion method and the system identification (SI) method, are used for model linearization.

The designed ASC system was validated by some initial tests, and following that, the tow tank tests were performed to determine the vehicle hull resistance and self-propulsion points. Based on the tow tank test data, a propulsion system model was built, and these results were validated by sea trials performed in Holyrood, Conception Bay South, NL. Using the sea trials' data, a state-space steering model for the ASC was identified based on the SI method.

Acknowledgements

I would like to express my sincere thanks to my supervisor Dr. Ralf Bachmayer for his support through the whole process of this thesis project, his suggestions and ideas were essential in the design of this first version ASC in the AOSL at Memorial University. At the lab, we had a very friendly ambiance in discussing all the academic topics. Brian Claus provided great suggestions in the communication system design part; Haibing Wang gave me some good advice in developing the mechanical structure of the system; Mingxi Zhou and Mohamud Hasan gave me a great assistance during the vehicle tow tank tests and open water tests. Specially, I want to give thanks to flow tank technician Trevor Clark for his cooperation in the resistance and self-propulsion tests. Many thanks to the lab project manager Neil Riggs for his arrangement for the transportation of the ASC, and Craig Legge, the technician at Holyrood, for his help in the ASC open water tests. Finally, I would like to thank my parents' love and my girlfriend's support during my master degree. I really enjoyed my master study and most importantly I learn a lot of knowledge through this process.

List of Tables

List of Figures

List of Abbreviations

AHRS	Attitude and Heading Reference System
AOSL	Autonomous Ocean Systems Laboratory
ASC	Autonomous Surface Craft
AUV	Autonomous Underwater Vehicle
CAN	Controller Area Network
CB	Center of Buoyancy
CG	Center of Gravity
CRC	Cyclic Redundancy Check
CTD	Conductivity-Temperature Depth Sensor
DIN	Deutsches Institut für Normung
DOF	Degrees of Freedom
GPIO	General Purpose Input Output
GUI	Graphic User Interface
MIMO	Multiple Input and Multiple Output
MISO	Multiple Input and Single Output
NMEA	National Marine Electronics Association
PIC	Peripheral Interface Controller
PRBS	Pseudorandom Binary Sequence
RC	Remote Control
RF	Radio Frequency
SI	System Identification
SNAME	Society of Naval Architects and Marine Engineers
SWATH	Small Waterplane Area Twin Hull
TTCAN	Time-Triggered CAN
USB	Universal Serial Bus
WS	Weather Station

Chapter 1

Introduction

1.1 Past ASC Developments and Applications

An Autonomous Surface Craft (ASC) or Unmanned Surface Vehicle (USV) is a type of marine robotic device that can operate autonomously or be remotely controlled in lakes, rivers and the oceans. With increasing interest in the ocean environment exploration and inland water area monitoring, various ASC prototypes have been proposed in the past 20 years. As an example, Figure 1.1 shows a catamaran-type ASC that the author worked on for the past two years in the Autonomous Ocean Systems Laboratory (AOSL) at Memorial University [1].

In the United States, the ASC prototype ARTEMIS was firstly introduced by the MIT[1] Sea Grant College Program in 1993, and in this design a scaled model (1/17) trawler boat, was produced mainly for the validation of the navigation and control systems [2] [3]. However, owing to its small size, ARTEMIS had limited endurance and could only perform a simple bathymetric survey within the Charles River in Boston. Subsequently, to increase the size and endurance, a kayak hull based ASC

[1] Massachusetts Institute of Technology

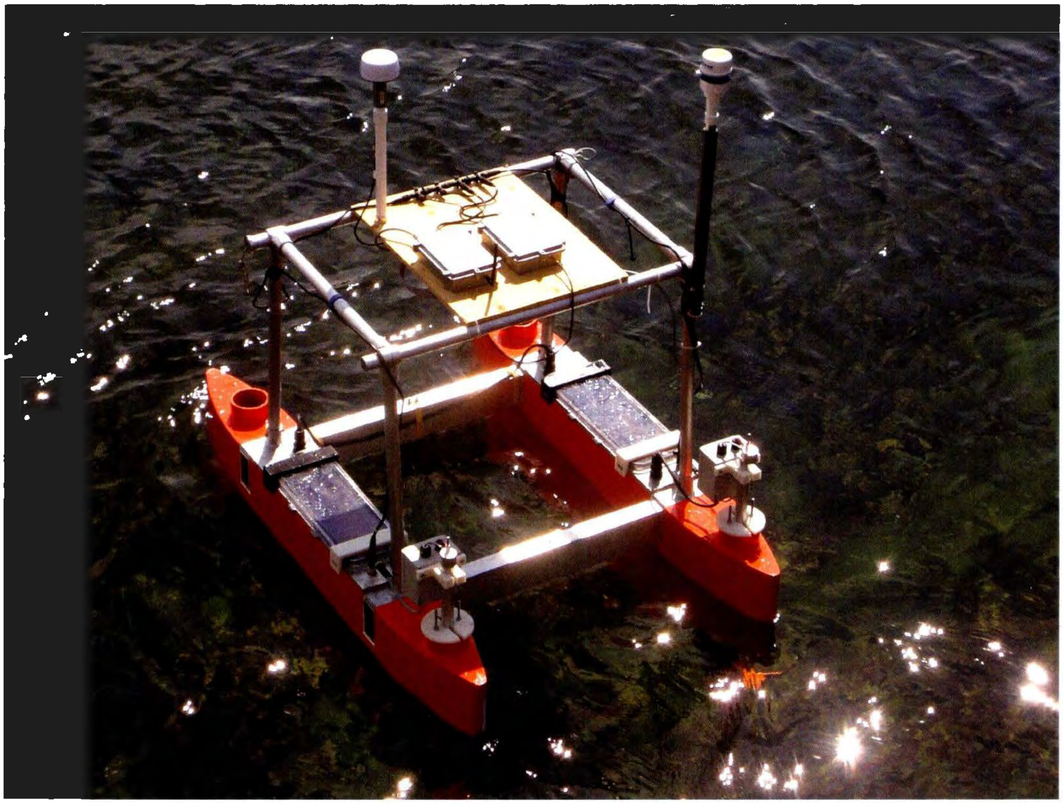

Figure 1.1: A catamaran-type ASC under development at Memorial University [1]

was proposed. Equipped with an acoustic tracking system, this kayak ASC could complete the task of tracking tagged fish in an open sea environment [4]. In 1996, a catamaran-shaped ASC, ACES [3], was developed to provide better roll stability and longer endurance compared with former designs. ACES used a gasoline engine for propulsion and batteries to power the on-board electronic systems. The mechanical structure that linked the two hulls was also the support for the sensors that were suitable for the hydrographic survey. Though the gasoline engine could give the vehicle satisfactory speed, pitch oscillations at high speeds affected the bathymetric measurements, which led to the modification of the entire mechanical system after the initial sea trials. During this overhaul, the vehicle was stabilized and outfitted with an electric propulsion system to enable better controllability [5]. Inspired by the MIT projects, many academic institutions in Europe began to develop their own ASCs in the late 1990's, and almost all the institutions preferred to use the catamaran-type ASC prototype for better roll stability and more payload capacity. Starting from 1998, the ASC MESSIN [6], designed using the Small Waterplane Area Twin Hull (SWATH) principle, was developed in Germany for ocean survey and human rescue. The MESSIN proposed an accurate navigation system in its operation, and the model based control algorithm allowed for the desired path-following in ocean exploration. Initiated in 1998, the project ASIMOV [7] was introduced to research the method for coordination between the ASC and an Autonomous Underwater Vehicle (AUV) in ocean data acquisition. In this project, a catamaran-type ASC Delfim [7], developed by Lisbon IST, was capable of building a fast data communication link with the AUV, and the Delfim could collect the ocean data independently based on its own onboard sensors. Following up this trend, the Charlie ASC sponsored by the Institute of Intelligent Systems for Automation, from the National Research Council of Italy (CNR-ISSIA) was specifically developed for Antarctica sea surface microlayer sampling [8]. Furthermore, the Springer ASC supported by the University of Plymouth UK, was used for environmental monitoring and pollutants tracking [9]. These early explorations formed an excellent basis for the accelerating improvement of the ASC in the past few years. In 2004, the kayak model based ASC SCOUTs were developed by the MIT Sea Grant to serve as a test platform for various research purposes [10]. These applications include the validation of multi-vehicle coordination between the ASCs or the ASC and AUV and target tracking control using multiple vehicles (Figure 1.2). In addition, equipped with a winch system and a Conductivity-Temperature Depth Sensor (CTD), SCOUTs could perform more sophisticated ocean monitoring tasks. Based on the Charlie ASC design, CNR-ISSIA proposed the SESAMO project with a new sea surface microlayer sampling mechanism for service in Antarctic coastal areas used to determine the interaction between the ocean and low atmosphere [11]. In Portugal, the catamaran-type ASC ROAZII was developed to perform basic bathymetric surveys and was mainly used for coastline shallow water land interface zones risk assessment [12].

Figure 1.2: Multi-ASC cooperations [10]

The annual International RoboBoat Competition hosted by AUVSI[1] [13] in the United States attracts students from all over the world to become involved in intelligent ASC development. Different from academic designs, the competing ASCs' hull types are more flexible and the developed system mainly focuses on the competition events; however, outfitted with the navigation, control and propulsion system, the ASCs gained excellent performance in water. The highlight of the competing ASCs is the successful integration of a vision system, enabling color identification functionality.

[1]The Association for Unmanned Vehicle Systems International

These designs validate that vision system based ASC prototypes are possible to be constructed in support of obstacle avoidance. Figure 1.3 shows a vision-based ASC example from Florida Atlantic University [13].

Figure 1.3: GUSS ASC with vision system from Florida Atlantic University [13]

In the military, ASCs were mainly used for coastal and harbor security and mine sweeping purposes. In 2007, the US Navy announced "The Navy Unmanned Surface Vehicle Master Plan", which detailed how the US Navy would catch up in the research of ASC in the following few years. In this report, the ASC's unique position in the construction of the naval network is indicated, since it serves as the communication interface between the land, the air and the underwater vehicles. Following this trend, some US companies began to develop conceptual ASCs for both commercial and military implementations. The semi-submersible ASC 6300C (or C-Hunter) from ASV Ltd., as shown in Figure 1.4, was produced for various ocean applications [14]. Its single hull small waterline feature provides excellent stability in the ocean, and equipped with side-scan sonar, CTD[1] and other sensors, guaranteed the use of the vehicle for various scientific tasks. The catamaran-type ASC C-CAT proposed by the same company was supposed to have the long-range communication capability as far as 8 km [15]. To further increase the endurance, developers started to look into the usage of renewable energies in the ocean, such as wind, waves and solar energy. Unmanned Ocean Vehicles Inc. proposed a single hull ASC with rigid sails and solar cells for onboard power; Emergent Space Technologies produced a mono hull solar powered ASC OASIS [16]; Liquid Robotics, on the other hand, provided a creative design that took advantage of wave energy. Though all these designs still remain at the level of system testing, they illustrate the potential use of renewable ocean energies as a primary or secondary energy source in future design for long range and long endurance ASCs [17].

In Australia, a solar powered catamaran-type ASC was developed in 2009 [18], and this vehicle was used for inland water quality and greenhouse gas monitoring. In 2011, McGill University introduced the catamaran-type ASC MARE [19], and the highlight of this vehicle is its usage of the air propeller as the propulsion system, which could perform the ocean surveying tasks with the minimum disturbance to the sea surface. The AOSL at Memorial University started to develop a SWATH-type ASC concept in 2010 for the task of underwater glider recovery [20]. Based on previous system design experience, a new multi-purpose catamaran-type ASC is designed (Figure 1.1). Special emphasis is given to robustness and operational capabilities in the coastal

[1] CTD is an essential instrument in physical oceanography to measure Conductivity, Temperature, and Depth.

Figure 1.4: C-HUNTER semi-submersible ASC from ASV Ltd. [14]

waters offshore Newfoundland, and since this ASC is a first prototype, it is used to develop the communication, control and power system architecture, and provide us with operational experience in the coastal waters of Newfoundland.

1.2 The ASC System Design Methods Comparison

Though various methods have been used in developing the communication and control system for previous ASCs, from a high-level view, all these systems can be generally divided into two categories: centralized and decentralized systems. Figure 1.5 shows a simplified structure of these two topologies. Each node is represented by a black box and each solid line represents the physical and information connection between the nodes. In the real world, the nodes are computers, microcontrollers or sensor units, and the information is exchanged by using different communication protocols. The left centralized system consists of 5 nodes, and node 1, as the main processing unit, has to handle all the information and functions transferred from different modules. In the right decentralized system, each node plays a relatively equal role in the information exchange and data processing [21].

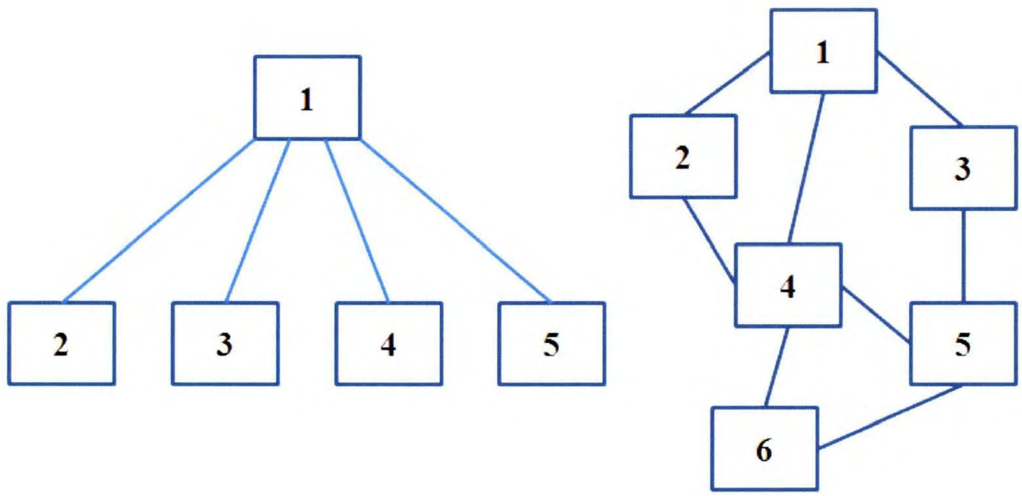

Figure 1.5: Centralized and decentralized system topology

Based on the proposed categorization and the simplified topology expression, it is possible to evaluate the two ASC system design methods [22]. The centralized topology is the most straightforward structure; all the accessory equipment is connected with the one core processing unit. Since all the data are gathered into one place, the main processing unit is capable of managing the whole system. A centralized structure is easy to implement and by designing robust and intelligent software for the main processing unit. it is possible to guarantee system security. This system design technique is widely used in the development of an ASC system; however, as not considering the fault-tolerance and extensibility of a system, the whole system is shut down even if a small fault occurs, since almost all data is centralized on the central processing unit.

An argument for centralized structure is that more reliable software can be designed to avoid the whole system shut-down; however, development will take more time, and an increase in software complexity can lead to difficulties maintaining and modifying the system. Regarding system extensibility, the centralized structure is limited by the main processing unit resources.

System fault-tolerance and extensibility can be acquired by implementing the decentralized system. In a decentralized system, each node has its own assigned task and is playing a relatively equal role for data processing and exchange, so when one or even a few nodes do not work properly, the rest of the system can still operate. The decentralized system extensibility is not restricted to the central processing unit, and additional node can join the original system without changing the existed programs and system structure. With multiple nodes working together, a decentralized system also tends to accomplish more sophisticated tasks than a centralized system. However, a decentralized system is hard to manage, because different nodes work separately.

1.3 Problem Statement

The size and weight of most existing ASCs are relatively small, so these ASCs are sensitive to environmental interferences including wind, current and wave. According to the literature review, many existing ASCs were designed for the inland water applications such as lakes, rivers and reservoirs [3]-[19], where weather is predictable and has little effect on the ASCs' proper operation. However, for the implementation of an ASC system in harsh ocean environment, especially offshore Newfoundland, the environmental situation has to be considered. Low temperature, strong wind and large currents and waves have a big effect on the working status of the ASC; for reliable ocean exploration, a robust ASC system design is needed.

The ASC system is required to carry different sensors in support of the environmental monitoring and measurement tasks, therefore a system structure that supports fast integration and flexible connection with the potential sensors is preferred. For reliable operation in the ocean, ASC system fault-tolerance is also necessary. Compared with the centralized system, it is more difficult to build a decentralized system, because each node inside the decentralized system has to be designed and programmed independently. However, to deploy an ASC for ocean exploration offshore Newfoundland, the ASC system reliability, fault-tolerance, extensibility and design difficulty are comprehensively considered. Eventually, a decentralized ASC system structure is chosen.

To tackle the downside of a decentralized system, a Controller Area Network (CAN) protocol based bus architecture is used. The proposed CAN-bus system can increase the decentralized system manageability, because all the nodes share the same physical transmission media for information transmission. The CAN network robustness is guaranteed by the CAN protocol defined error detection mechanism.

1.4 Thesis Outline

Chapter 1

Past development of the ASC is discussed, and based on the literature review, two ASC system design methods are compared. The challenge of developing a robust ASC for deployment in the harsh ocean environment, especially offshore Newfoundland, is introduced.

Chapter 2

An overview of the design considerations of the developed ASC system is provided. A general introduction of the CAN, NMEA 2000 standard and Time-Triggered CAN protocol is presented. The details of how the on-board communication system is built are introduced, and the detailed program design methods for different CAN nodes and a software Graphic User Interface (GUI) are provided.

Chapter 3

Generation of a simplified 3 Degrees of Freedom (DOF) nonlinear model for describing the dynamic motion of the ASC. Two linearization methods are applied to achieve the linearized ASC model.

Chapter 4

Evaluation of the proper functionality of the ASC system by some initial tests. Tow tank tests have been performed to get the ASC hull resistance coefficient and self-propulsion points. Based on the tow tank test results, a propulsion model is generated. To validate this model, sea trials have been carried out in Holyrood, Conception Bay, Newfoundland. The comparison of these results are presented. Based on the sea trails data, a linear ASC steering model is identified.

Chapter 5

Conclusions and a description of future works.

Chapter 2

The Autonomous Surface Craft System Design

2.1 The ASC System Design Overview

In [20], a SWATH-type ASC development concept was developed in the AOSL at Memorial University for the task of underwater gliders recovery. Based on this previous system design experience, the AOSL started to develop a new catamaran-type multi-purpose ASC in 2010. This multi-purpose ASC is designed to be robust enough for operation in the coastal waters of Newfoundland and Labrador, with the principal tasks to collect oceanographic data and serve as a surface gateway for underwater vehicles assisting them with communication, navigation and control.

In this part, an overview of this developed ASC system is provided. Figure 2.1 shows an original design of the AOSL ASC. In this design a twin hull catamaran-type ASC is built. Two round through-holes from top to bottom of the hull are located on the front and rear part of the vehicle. The aft through-holes are used for holding the propulsion system, while the front ones are used for connecting the bottom sensors to the main communication system. Inside each hull, there is space for storage of batteries and other electrical components, and each hull is sealed with the transparent plastic hatch cover. The two hulls are ruggedly connected by two aluminum beams. On top of the ASC hulls is superstructure tubing, used for mounting antennas and necessary electric components.

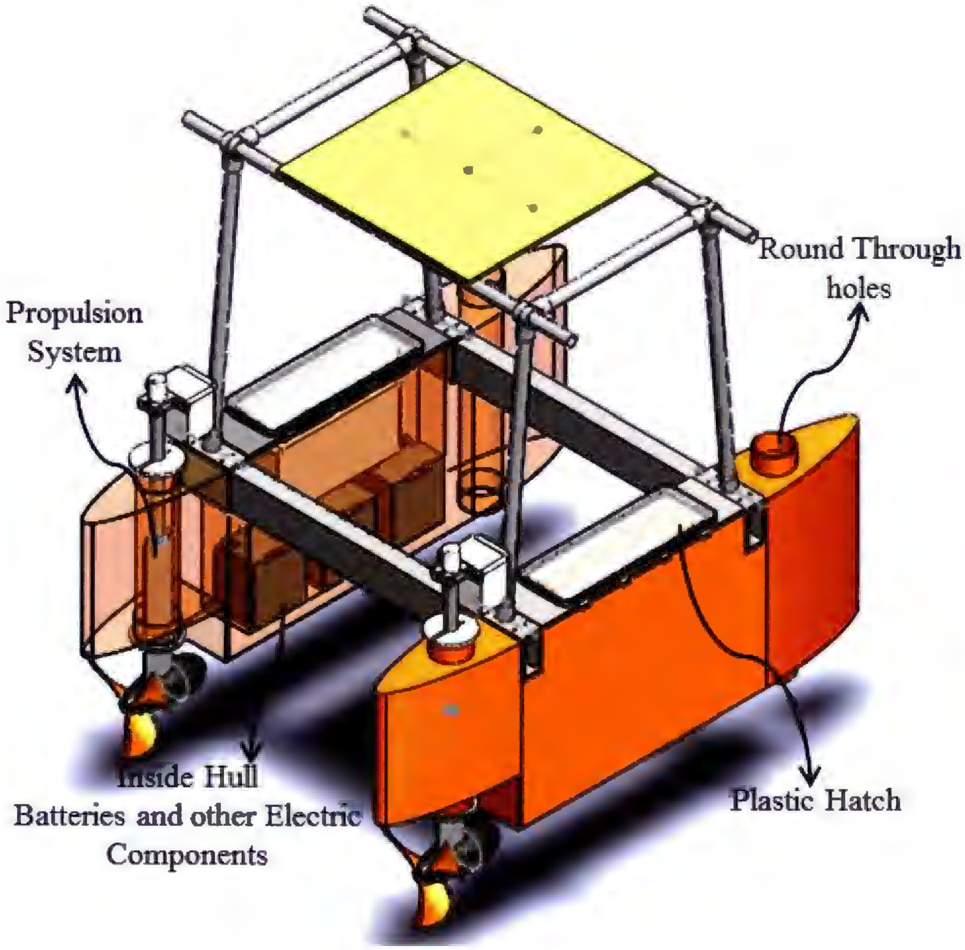

Figure 2.1: The AOSL ASC original design

Figure 2.2 shows the final realization of the AOSL ASC system. This catamaran-type ASC is driven by two independently controlled electric motors installed at the rear part of each hull. Though there is no rudder in this ASC design, the vehicle can be steered by applying differential thrust from the propulsion system. This ASC measures 1.5 m in length and 1 m in width, and the draft of the vehicle is around 0.37 m under testing and working conditions. With six batteries and the superstructure mounted on the vehicle, the total weight is 146 kg. The on-board communication system is built using the Controller Area Network (CAN) protocol, and based on the designed decentralized system structure, four CAN modules were developed independently (labeled from 1 to 4 in Figure 2.2) to work on this CAN network which will be described in more detail in the electrical system design section. A weather station (WS) is installed to measure the wind data, temperature and barometric pressure, and a CAN node that integrates the Global Positioning System (GPS) and Attitude and Heading Reference System (AHRS) is used to provide accurate ASC navigation information. In addition, a 900 MHz wireless modem is integrated to the system to enable on-board sensor data and supervisory commands exchange between the ASC and the dock-side computer. As shown in Figure 2.2, the ASC features the distributed communication system structure and in total four separate CAN modules were developed. Since each CAN module has its own assigned task, they have to be developed with their own programs. At the same time, in order to increase system manageability and make all nodes work in an orderly way, the Time Reference Message (TRM) time synchronization mechanism is introduced. Moreover, a GUI software package is developed on the dock-side computer to work with the wireless communication system for sensor data display and ASC control. A discussion of programs and software realization details is provided in the program development section.

A summary of the main specifications of the developed ASC system is provided in Table 2.1.

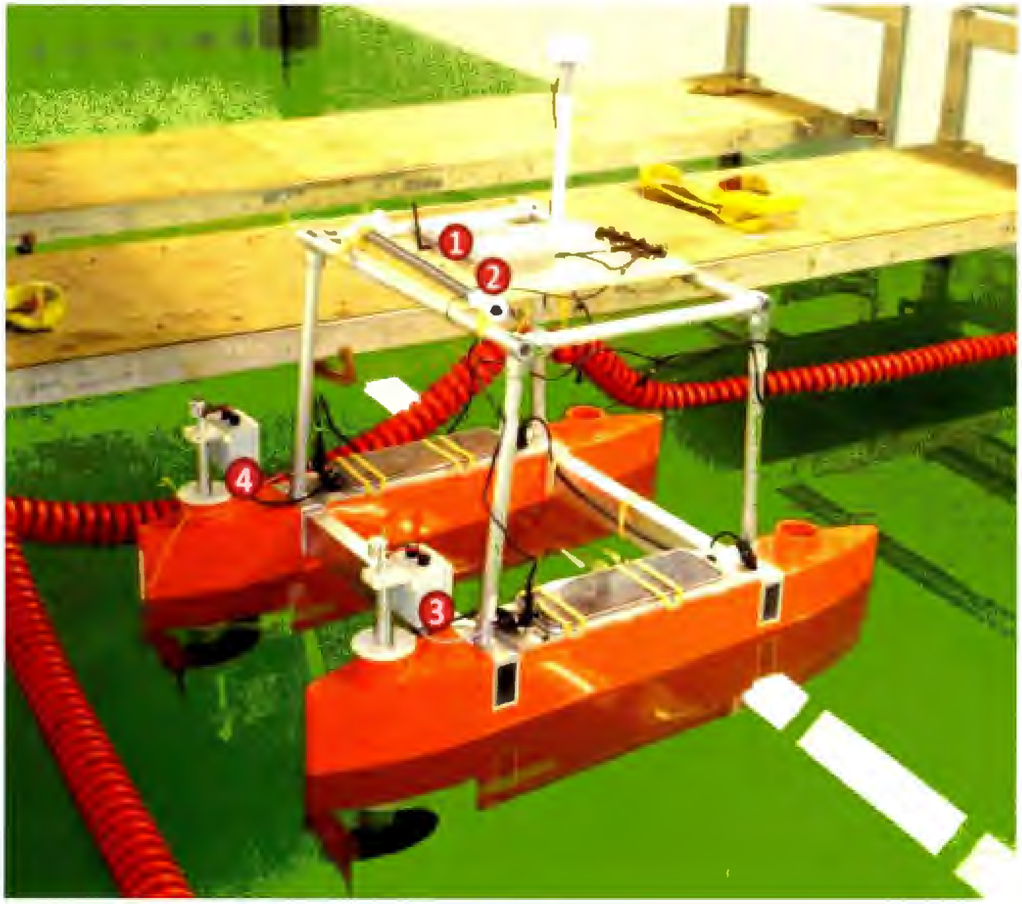

Figure 2.2: The AOSL ASC final realization

2.2 Electrical System Design

The proposed on-board distributed communication and control network is based on the classic CAN protocol. However, since the NMEA 2000[1] standard is used for the communication with the WS and the Time Reference Message (TRM) based system synchronization method is inspired by the Time-Triggered CAN (TTCAN), introductory sections of these protocols are provided. The electrical components used to build the on-board electrical system are introduced, and the concentration is on how to use these off-the-shelf components to build separate CAN nodes. The schematics

[1]The NMEA 2000 protocol is based on the extended mode CAN messages.

and the final realization of these developed CAN nodes are shown.

Table 2.1: The ASC main specifications

Length	1.5 m
Width	1.0 m
Propeller offset from the longitudinal centreline of the ASC	0.5 m
Hull height	0.5 m
Superstructure height	0.66 m
Total weight	146 kg
Draft	0.37 m
Speed	0.4 to 1.0 m/s
On-board Communication	CAN-bus
External Communication	900 MHz wireless
Operating mode	Manul/Autonomous

2.2.1 Controller Area Network (CAN)

The Controller Area Network (CAN) was designed by the German company Bosch in 1986, and it was originally used in the construction of the communication and control system for automobiles [24]. A typical system topology using the CAN protocol is shown in Figure 2.3, and in this system a number of CAN nodes are connected to the two-wire CAN-bus system that is terminated by two 120Ω resistors. As a multi-master communication protocol, all the CAN nodes can freely access the bus to send and receive the messages. As discussed in Chapter 1, the CAN-bus system can be regarded as a decentralized system. A general review of this protocol is provided in this section, and the main focus is on the usage of this protocol to build the ASC distributed communication and control system. However, interested readers can refer to the book [24] for more details of the CAN protocol.

CAN-bus is based on a message-oriented communication mechanism where a CAN message is broadcasted by one CAN node with a unique identifier (ID) number. All the other connected CAN modules can hear the transmitted message but will only accept the message with the specific ID they define. The CAN message priority is determined by its ID, with a lower ID number having a higher priority. When multiple CAN modules try to access the CAN-bus at the same time, the higher priority message will be transmitted firstly, and the other messages' transmission will be delayed. This loss-free bus arbitration mechanism guarantees a robust CAN message transmission link where no messages are lost due to transmission conflicts, and high priority messages are transmitted with the shortest latency. As shown in Table 2.2, there are four kinds of CAN messages (or frames) defined in the CAN standard. Among these frames, the Data Frame and the Remote Frame are used in the construction of this distributed CAN network.

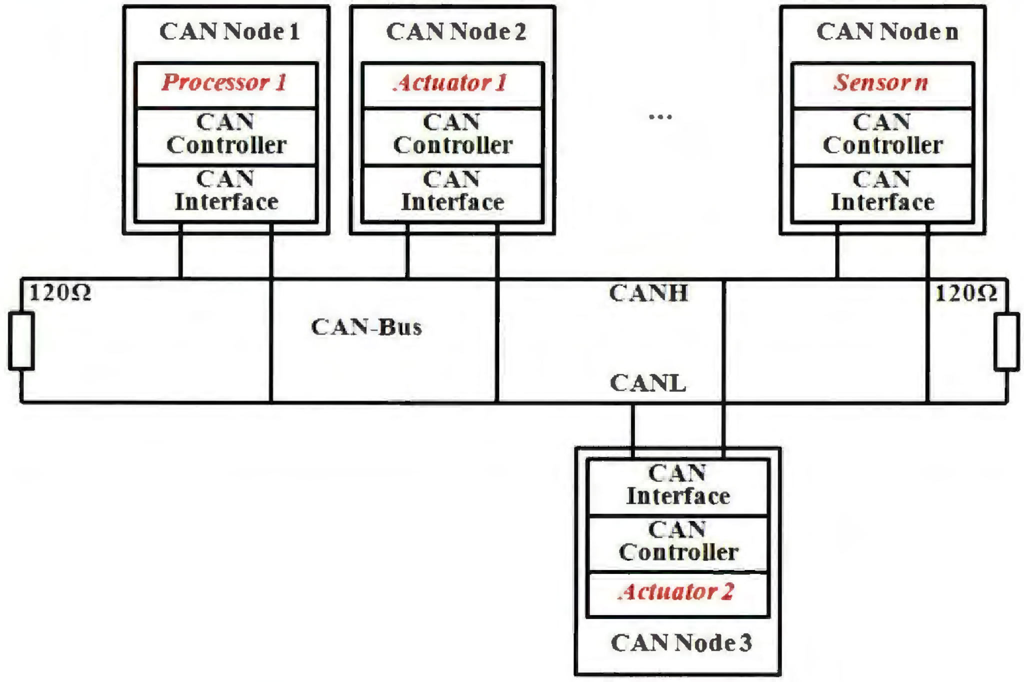

Figure 2.3: A typical CAN-bus network

The classic CAN protocol defines two working modes for data transmission: the standard mode, or CAN 2.0A, and extended mode, or CAN 2.0B. The standard CAN mode is used to build the main on-board communication network, while the NMEA 2000 standard, which implements the CAN 2.0B as low level communication protocol is used for the communication with the WS. The main difference between the standard and extended CAN messages is ID number length. The standard data message implements an 11-bit length ID; while the extended message ID is 29-bits long. The length of ID number determines the maximum number of messages that can be defined and used in designing a CAN network.

Table 2.2: Four kinds of CAN frames

Frame types	Description
Data Frame	Frame for data transmission
Remote Frame	Frame for request of data frame
Error Frame	Frame for issuing the error occured at one CAN node
Overload Frame	Frame for delay of the next transmitted message

Remote Frame is used for inquiry about a specific ID Data Frame. In a remote frame transmission, no data field is included; however, the CAN node that successfully receives the remote message will respond with a data frame with the same ID as the remote frame.

On-board communication system robustness can be guaranteed by the CAN protocol defined error detection mechanism. Each CAN node can use four error detection methods (Table 2.3).

According to the previous analysis, the reliable CAN-bus based communication system structure implemented on the developed ASC is shown in Figure 2.4. The center line indicates the CAN-bus trunk line. Four CAN modules are designed separately to work on this system. These CAN modules include two motor controller modules, and they are responsible for the control of the propulsion system. The navigation module is used for gathering the information from the GPS and the AHRS to assist with the vehicle navigation, and the controller module is used for the wireless communication and the communication with the WS. Each CAN node implements a combination of CAN controller chip and transceiver chip together as the CAN interface.

Table 2.3: CAN-bus error detection mechanisms

Error detection method	Description
Bit check	Each CAN node checks if the transmitted bus level and the actual bus level is different
Frame check	Each CAN node checks the fixed part of the transmitted frames
Cyclic redundancy check	Enable the receivers to verify the integrity of the whole transmitted data
Acknowledgement check	Check if there is response for the sending message in the ACK part of CAN message format

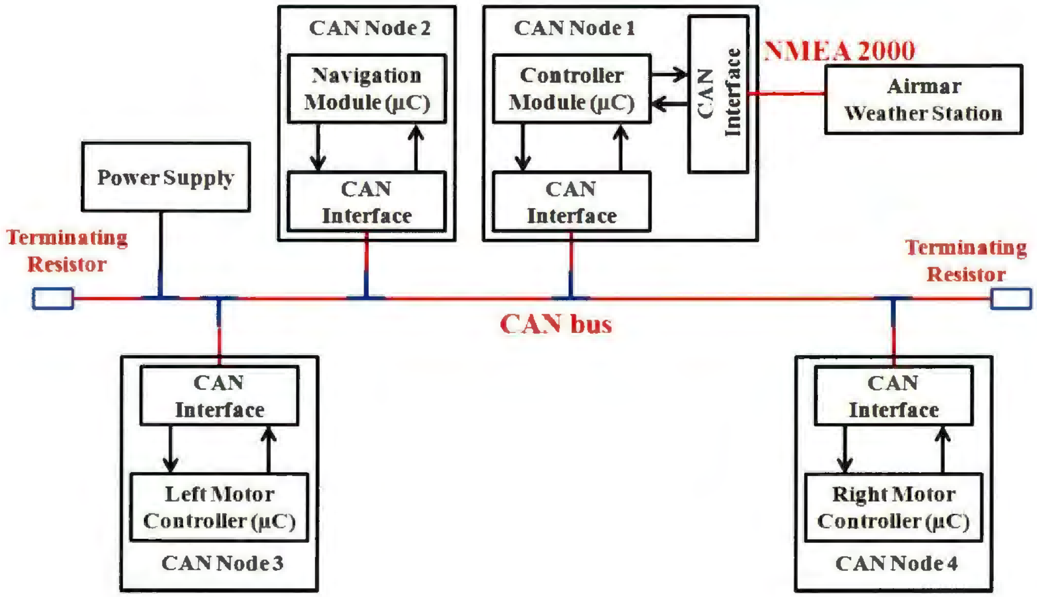

Figure 2.4: CAN-bus based decentralized ASC communication and control system structure

Figure 2.5 provides a detailed layout of the proposed CAN network implemented on the ASC. As shown in the figure, the CAN-bus runs through the whole system with four CAN modules distributed at different locations on the ASC. Two motor controller modules are fixed at the rear part of each hull, and both the controller module and the navigation module are installed on the superstructure. The main characteristics of this developed ASC CAN-bus based communication system are summarized in Table 2.4.

Table 2.4: CAN-bus application characteristics

Characteristics	Description
Topology	Bus topology
Number of nodes	Four
Data transmission bit rate	1Mbps
Data format	Standard and NMEA 2000

2.2.2 NMEA 2000

The NMEA 2000 standard was introduced by the National Marine Electronics Association (NMEA) in 2001 [25]. NMEA 2000 implements CAN 2.0B as a low level communication protocol, and it is mainly used for building the control and communication system for marine vehicles. The NMEA 2000 standard provides much faster data transmission rate than the NMEA 0183[1] standard, and since it is based on CAN-bus, a reliable and extensible bus architecture can be achieved. As a high-level communication protocol, NMEA 2000 develops a more advanced message identification mechanism. The 29-bits identifier (ID) number is divided into several parts for representation of different characteristics of the transmitted messages. In Table 2.5, the message types, priority and update rate are specified inside the Parameter Group (PG), which is the ID number of corresponding extended CAN messages.

[1]NMEA 0183 is another widely implemented serial protocol that is used for marine electronic devices communication

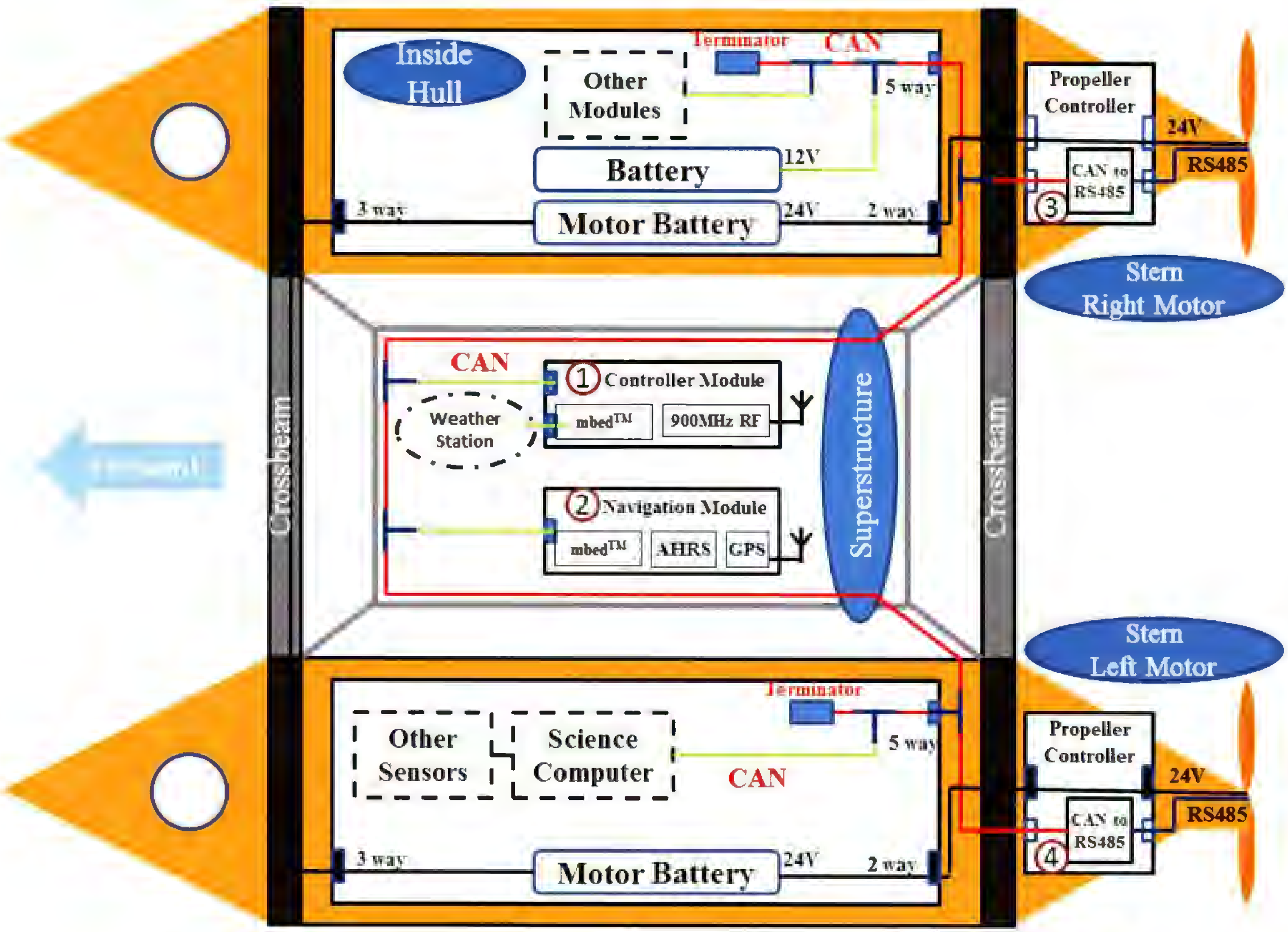

Figure 2.5: The ASC system schematic layout

Table 2.5: Parameter Group

Name	Description
PGN	Defined by the NMEA committee for identification of different type messages and this part could also be used for company proprietary transmitting messages
Destination	Define if the message is global or addressed
Default priority	0 to 7 priority range
Update rate	Define how often the message is transmitted
Query support	Define if the transmitted message will respond to request messages
Single frame	Define if the transmitted message is single-frame or multi-frame
Acknowledgement	Specify if the reply is needed after receiving this message

In this ASC on-board communication system, the NMEA 2000 standard is used only for communication with the WS. Based on the open source project, which was posted on the blogger [23] and developed by Keversoft B. V., the following useful information as shown in Table 2.6 has been successfully requested from the WS.

Table 2.6: Used NMEA2000 messages

PGN number	Description
PGN 127250	Vessel Heading
PGN 127251	Rate of Turn
PGN 129025	GPS Position
PGN 129026	Course Over Ground Speed Over Ground
PGN 129033	Time and Date
PGN 130306	Wind Data
PGN 130310	Environmental Parameters

The classic CAN protocol does not define the physical layer cables or connectors, so commonly a twisted pair CAN wiring system is applied. However, in this application the CAN-bus has to run through the ASC including the superstructure, and the wiring system is exposed to rain and water, therefore, a robust and waterproof wiring system is needed. The NMEA 2000 standard recommended Micro-type cables and connectors to help solve this problem. On the one hand, NMEA 2000 low level communication is based on CAN protocol, so CAN messages can be transmitted inside this wiring system; on the other hand, since the NMEA 2000 is designed for marine vehicles, the cables and connectors follow Ingress Protection Rating 67 (IP67) waterproof standard. Figure 2.6 shows the implemented NMEA 2000 Micro-type 5-pin cables and connectors on the designed ASC. The definition of each pin is provided in Table 2.7.

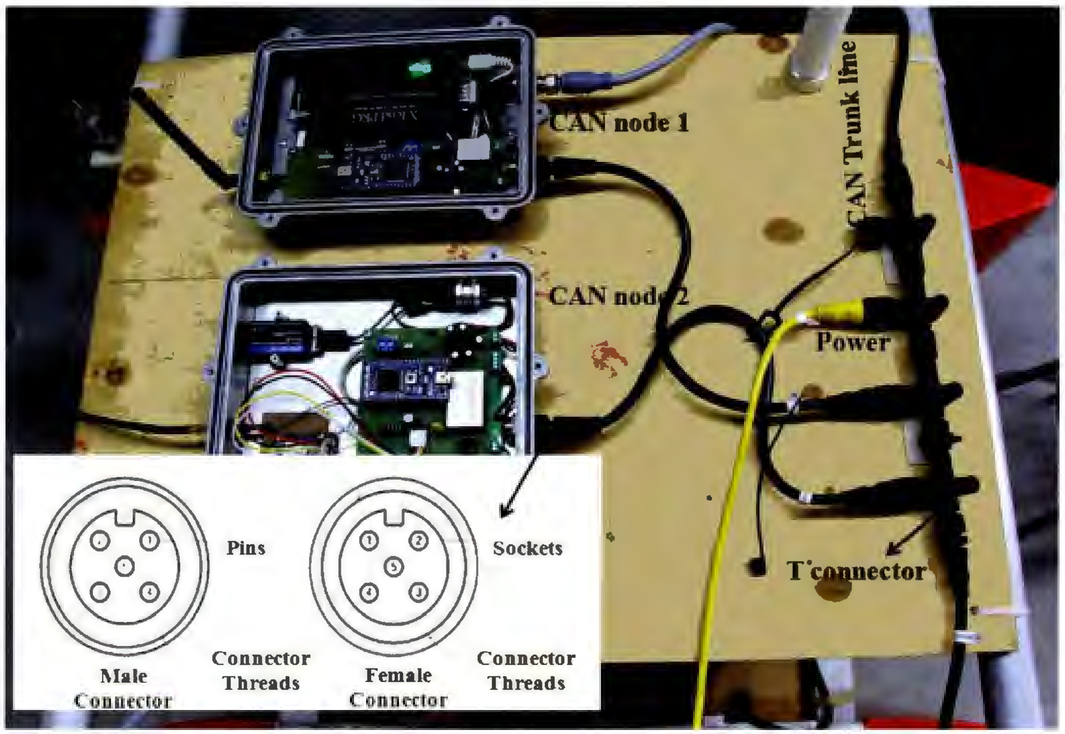

Figure 2.6: On-board communication system based on NMEA 2000 cables and connectors

A couple of advantages brought by the NMEA 2000 standard are:

- The NMEA 2000 cable consists of two pairs of shielded-twisted wires, where one pair is for data transmission, and the other pair is for power transmission.

The remaining is one shield wire helps increase the internal signal resistance to interference as well as reduce the RF emission.

- The NMEA 2000 cables and connectors are designed for marine usage, so the waterproofness and robustness in the ocean are guaranteed.
- As shown in Figure 2.6, CAN nodes can be connected with the CAN trunk line through the use of three-port "T" connectors, and they can get access to CAN communication network and power at the same time which brings convenience for system extension.

Table 2.7: NMEA 2000 cables and connectors pin definition

Pin number	Definition
Pin 1	Shield wire
Pin 2	+V
Pin 3	-V
Pin 4	CANH
Pin 5	CANL

2.2.3 Time-Triggered CAN (TTCAN)

Time-Triggered CAN (TTCAN) is developed based on the classic CAN protocol, and it is used to support the time-deterministic data-transmission applications. In a classic CAN network, it is possible that different nodes start to transmit their messages simultaneously and when this happens the message with higher priority will be transmitted first. Though this bus arbitration mechanism ensures the high priority CAN messages are transmitted with the minimum latency, it is difficult to guarantee that low priority CAN messages can meet their transmission deadlines under different bus load conditions. TTCAN is introduced to solve this problem, and in this protocol, to assist with the time scheduled behaviour of the CAN-bus application, a periodically transmitted reference message is used to trigger the whole system to work under the same time step. The reference message indicates the start of one cycle, and each cycle is divided into three parts for different messages' transmission. The definition of the divided time windows and the specific messages that are transmitted within each window are provided in Table 2.8.

Table 2.8: TTCAN time windows

TTCAN windows	Description
Exclusive window	Time slot for transmitting the message without competition for the usage of the CAN-bus
Arbitrating window	Time slot for general CAN messages, bus arbitration works
Free window	Time slot for future extension

Inspired by the TTCAN, a Time Reference Message (TRM) based mechanism is developed to add the time scheduled behaviour to the original CAN network. In this method, the TRM is a standard data frame that contains UTC time information from the GPS module (inside navigation module as shown in Figure 2.5). The GPS data update rate is configured to be 1 Hz, so the TRM is transmitted on the CAN-bus every second. The TRM is received by all the connected CAN nodes, and it indicates the start of one unit cycle. As shown in Figure 2.7, node 1 and 2 only transmit their messages after the reception of the TRM, and the remaining time window is free for other messages transmission.

A couple of advantages have been brought by the introduced TRM mechanism.

- Add the time deterministic feature to the classic CAN-bus.
- TRM triggers the system to work under the UTC time, and it increases the system manageability.

- TRM can be regarded as a decentralized system time synchronization method, while no additional time line is needed which simplifies the system connection.

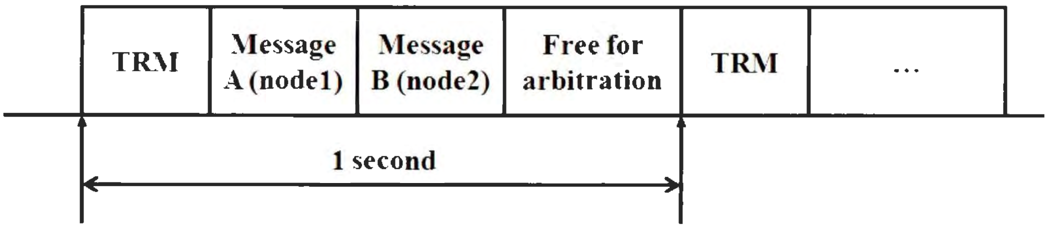

Figure 2.7: TRM mechanism unit cycle organization

2.2.4 Electrical Components

A general introduction of the implemented electrical components on the ASC system is provided.

2.2.4.1 Airmar PB200 Weather Station

When the ASC is operating in the ocean, it is important that the vehicle has access to environmental information of the area around it. Environmental information can assist the vehicle for analysis of its operating status, and it can also help the ASC to plan a safe moving path to avoid damaging environmental interferences.

Figure 2.8 shows the Airmar PB200 weather station. This weather sensor has a waterproof housing and is resistant to sunlight and chemicals [27]. As a compact design, the sensor is 130 mm high and 72 mm diameter with a mass of 285 grams. The PB200 sensor can measure wind speed and wind direction using its four ultrasonic transducers, located on top of the wind channel. Two transducers work together to measure the wind speed in that direction. As depicted in Figure 2.8, each transducer takes turns to transmit and receive the signal. The flowing air through the wind channel will affect the signal transmission time between the two transducers, and by measuring this time changes in the wind direction and wind speed can be calculated.

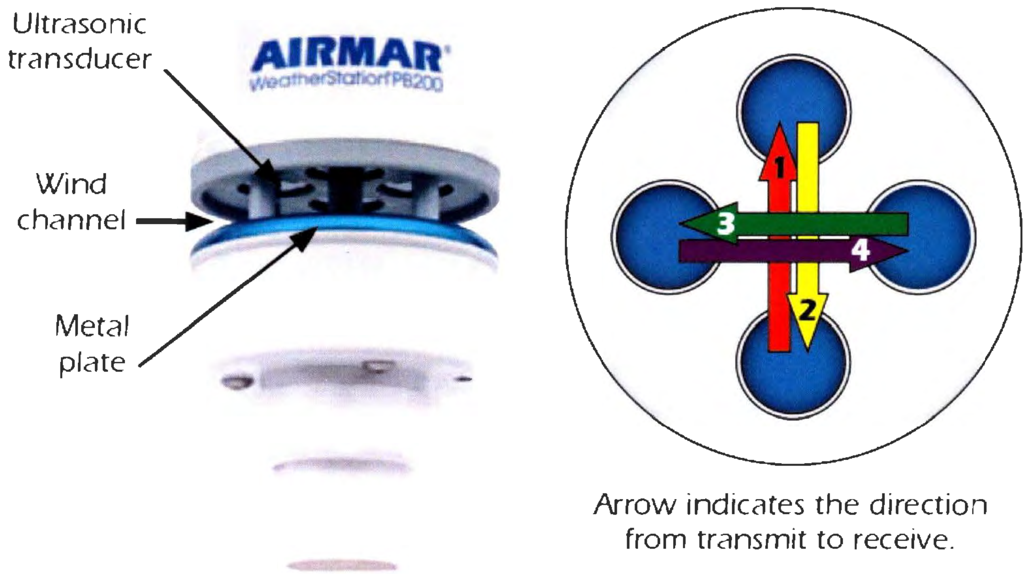

Figure 2.8: Airmar PB200 Weather Station and ultrasonic transducers (PB200 User Manual)

The PB200 weather station also integrates a temperature sensor, a barometric pressure sensor, a three-axis solid-state compass, a three axis accelerometer, a yaw rate gyro and a GPS module. Therefore, all required vehicle data can be requested from this sensor. The specific data measurement accuracy and range are detailed in Table 2.9. Though two interfaces are available in this sensor (NMEA 0183 and NMEA 2000), the NMEA 2000 standard is used as it provides faster data transmission rate. As shown in Figure 2.5, WS is connected with the controller CAN node.

Airmar's WeatherCaster software has been used for the initial test of the WS. After the basic functions are tested and every sensor is verified to work properly, the WS has been configured to transmit upon the reception of the NMEA 2000 standard defined request message.

Table 2.9: PB200 weather station specifications (PB200 User Manual)

Wind speed range	0 to 80 knots
Wind speed resolution	0.1 knot
Wind speed accuracy	1 to 2 knots (5 knots in wet condition)
Wind direction range	0 to 360°
Wind direction resolution	0.1°
Wind direction accuracy	2 to 5° (8° in wet condition)
Compass accuracy	1 to 2°
Rate of turn range	0 to 70° per second
Rate of turn accuracy	1° per second
Pitch and roll range	±50°
Pitch and roll accuracy	<1°
Air temperature range	-25 to +55 °C
Air temperature resolution	0.1 °C
Air temperature accuracy	±1 °C
Barometric pressure range	850 to 1150 mbar
Barometric pressure resolution	0.1 mbar
Barometric pressure accuracy	±2 mbar
GPS position accuracy	3 m

2.2.4.2 Attitude and Heading Reference System (AHRS)

The attitude and heading reference system (AHRS) is widely used in unmanned systems to support vehicle navigation. In general, an AHRS can provide inertial measurements including acceleration, angular rate and magnetic field, and combined with the GPS module, the AHRS can aid the GPS for better estimation of the unmanned system location and orientation. In this design, though we already have the AHRS function in the PB200 WS, an additional AHRS sensor is integrated into the system. There are two reasons for doing this: (1) redundant sensors can supply the measurements for the same physical variables, so the results can be used for data validation; (2) when one sensor fails to work, the remaining AHRS module can still provide the system with the necessary information.

The Microstrain 3DM-GX3-25 sensor (Figure 2.9) has been used. This lightweight

AHRS integrates a triaxial accelerometer, a triaxial gyro, a triaxial magnetometer, a temperature sensor and a processing unit with a data fusion algorithm. Therefore, fully temperature compensated acceleration, angular rate and magnetic heading data are available in this small sensor unit.

Figure 2.9: The Microstrain 3DM-GX3-25 AHRS sensor (3DM-GX3-25 User Manual)

Table 2.10: Acceleration and Angular Rate messages from the 3DM-GX3 (3DM-GX3-25 User Manual)

Command:	
Byte 1	0xC2
Response:	
Byte 1	OxC2
Bytes 2-5	Acceleration X (IEEE-754 Floating Point)
Bytes 6-9	Acceleration Y (IEEE-754 Floating Point)
Bytes 10-13	Acceleration Z (IEEE-754 Floating Point)
Bytes 14-17	Angular Rate X (IEEE-754 Floating Point)
Bytes 18-21	Angular Rate Y (IEEE-754 Floating Point)
Bytes 22-25	Angular Rate Z (IEEE-754 Floating Point)
Bytes 26-29	Timer
Bytes 30-31	Checksum

Communication with the 3DM-GX3 is based on the RS-232 serial interface, and the default data transmission rate is 115200bps. Specific sensor data can be requested by a connected microcontroller by issuing the required command. Table 2.10 shows an example where the AHRS outputs acceleration and angular rate messages upon the reception of the 0xC2 command. It is shown that the response messages from the AHRS include in the first byte a reproduction of the sending command, and after that are the three axis acceleration and angular rate data, which are represented using the IEEE-754 standard[1]. Based on this data transmission mechanism, the data including the acceleration, angular rate, the magnetometer data and the rotation matrix are requested and logged in the navigation module microcontroller (Figure 2.4).

2.2.4.3 Global Positioning System (GPS)

In order to acquire the accurate location of the ASC in the ocean, the Global Positioning System (GPS) is needed. There are many off-the-shelf GPS receivers available in the market, and some recently developed differential GPS modules can provide the distance accuracy within centimetres; however, in this design, the cost and desired accuracy balance is considered. Since the GPS and AHRS module will be sealed into a waterproof enclosure, an external antenna is desired for receiving the GPS signal from the satellites. Therefore, as shown in Figure 2.10, the SeeedStudio Grove-GPS (originally equipped with a patch antenna) becomes our final choice. Grove-GPS integrates the cost-efficient NEO-6M GPS receiver chip from u-blox Inc., and the UFL receptacle connector on-board enables the external antenna connection.

According to [29], the implemented NEO-6M stand-alone GPS receiver is supported by the high performance u-blox 6 positioning engine. The implemented Grove-GPS board features the NEO-6M UART serial interface for communication with the microcontrollers, and the output message voltage is regulated by on-board regulator to be compatible with most processors' voltage level. The cold start time for NEO-6M GPS is within 27 seconds, and it has the horizontal position accuracy of 2.5 meters.

[1] Use four data bytes to represent a floating point number.

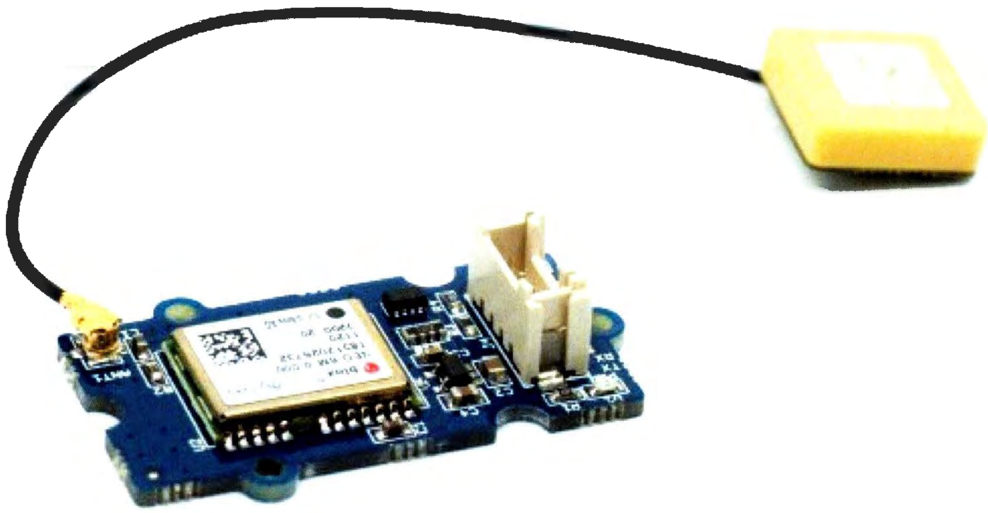

Figure 2.10: The Grove-GPS module (SeeedStudio Inc.)

The GPS data update rate is configurable from 1 to 5 Hz, which is good for fast update applications. The time-pulse signal is available on the "TIMEPULSE" pin of the NEO-6M chip, and its frequency is configurable from 0.25 to 1 kHz. The velocity accuracy is within 0.1 m/s, and the heading accuracy is less than 0.5 degrees.

The GAA-005 Marine GPS antenna has been chosen as the external GPS antenna. This antenna has a waterproof enclosure, and its working voltage range is from 2.2 to 5.0 V. The connected coaxial cable length is 30 m maximum, and this waterproof cable connects the GPS signal to the navigation CAN module (Figure 2.5).

The messages from the Grove-GPS are transmitted according to the NMEA 0183 standard, so the GPS data are included in the transmitted string. As shown in Table 2.11, the implemented GPRMC message string is provided, and inside this string the GPS data including UTC time, location, speed and course are provided. In this application, the GPS module is configured to update at 1 Hz.

Table 2.11: GPRMC message from the Ublox GPS module (Grove-GPS User Manual)

Field number	Example	Description
0	$GPRMC	RMC message header
1	083559.00	UTC time,hhmmss.ss
2	A	Status, V=data not valid, A=Data valid
3	4717.11437	Latitude, degrees=47, minutes=17.11437
4	N	Hemisphere N=north, S=south
5	00833.91522	Longitude, degrees=8, minutes=33.91522
6	E	E=east, W=west
7	0.004	Speed over ground, knots
8	77.52	Course over ground, degrees
9	091202	Date in day, month and year, ddmmyy
10	-	Reserved
11	-	Reserved
12	-	Reserved
13	*57	Checksum
14	-	Carriage return and line feed

2.2.4.4 Wireless Modem

To build a long range wireless communication link, the Digi International Inc. XTend-PKG 900 MHz RF modem has been used. Through this wireless communication link, the vehicle supervisory commands and important ASC operation status information can be exchanged between the ASC and the dock-side computer.

The XTend-PKG wireless modem features a long range signal transmission. The data transmission range is dependant on a couple of factors, such as the data transmission rate and the signal output power. The specific modem settings can be configured using the X-CTU software through the RS232 interface. In this application, to construct a reliable wireless link, both wireless modems are configured to work at "Multi-Transmit" mode. In "Multi-Transmit" mode, messages are retransmitted to guarantee the successful data transmission. To acquire the acceptable communication range as well as RF data rate, the two wireless modems are set to work at a one Watt power level, and the RF transmission rate is configured to be 115,200 bps. The modem was tested to work as far as 200 metres. However, according to the manual, when the transmission power is set to one Watt together with a high gain antenna, the outdoor RF line-of-sight communication range is up to 32 km (115,200 bps throughput data rate).

2.2.4.5 Microcontrollers

The mbed^{TM} microcontroller The mbed^{TM} microcontroller is designed for fast and reliable prototyping tasks. Its on-board processing unit implements the powerful 32-bit ARM[1] Cortex-M3 Core microprocessor NXP LPC1768, and it has the maximum processing speed of 96 MHz. As shown in Figure 2.11, the mbed^{TM} development board includes many useful resources including the Ethernet, USB, SPI, I2C, UART, CAN, PWM and ADCs.

The mbed^{TM} microcontroller can be powered by the Universal Serial Bus (USB), and the nominal current consumption is less than 100 mA. Each General Purpose Input Output (GPIO) pin is capable of driving up to 40 mA peripheral circuits with the total driving capability of 400 mA. The mbed^{TM} microcontroller uses 3.3 V logic but it can handle 5 V input signals. The mbed^{TM} microcontroller program development environment is based on an online compiler tool [28]. This online tool supplies the necessary libraries and functions for code development. Each mbed^{TM} microcontroller user has his own code development workspace, and all the developed code can be saved online for further adjustment.

As shown in Figure 2.5, since the mbed^{TM} integrates two CAN interfaces, it can be used to connect with the main CAN network on the one side, while it can also be connected to WS on the other side. This feature enables the mbed^{TM} microcontroller

[1] Advanced RISC Machine

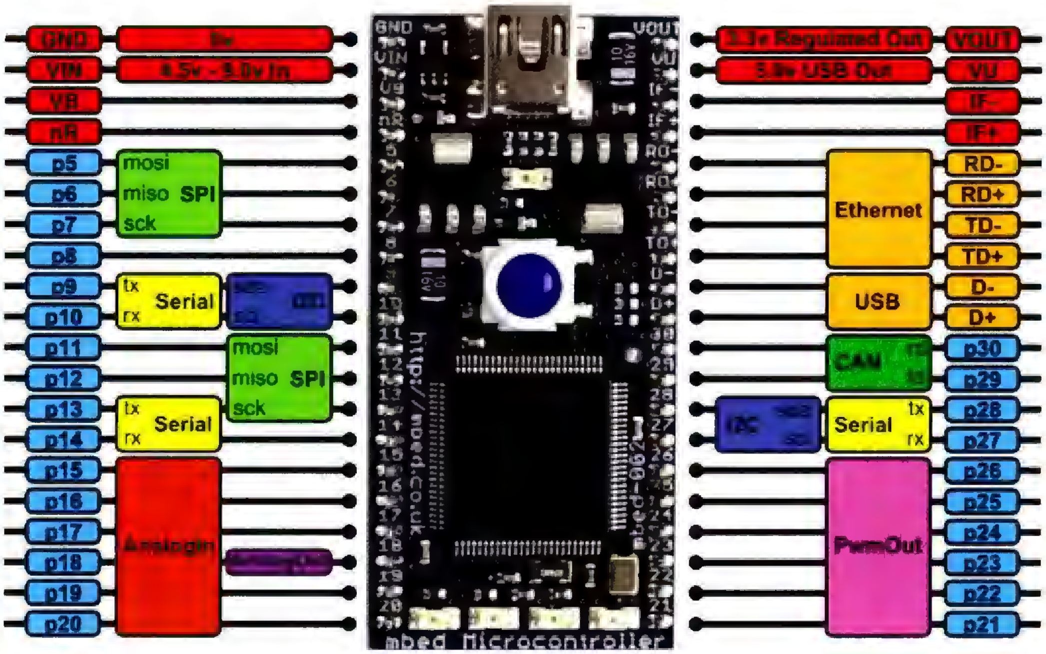

Figure 2.11: The mbed^{TM} microcontroller (http://mbed.org)

to serve as the main processor unit in the CAN controller module. Owing to its relatively fast processing speed and low power consumption, the mbed^{TM} microcontroller is also used with the CAN navigation module (Figure 2.5).

The PIC microcontroller The communication with the propulsion system is restricted to the RS485 serial interfaces, so a CAN to RS485 converter is required. In this design, the 28-pin PIC microcontroller-based single-board computer SBC28PC, shown in Figure 2.12, is implemented. This development board features a compact dimension of 58 mm x 54 mm, and it integrates the CAN and RS485 interfaces. The 8-pin socket is where the CAN driver chip is inserted and when CAN functionality is enabled the CAN signals are available on the 5-pin terminal block connectors.

The CAN to RS485 message conversion is done by the program running in the SBC28PC development board (Appendix A.3). Using this program, the SBC28PC board can inquire about the motor information, store the information, and when it

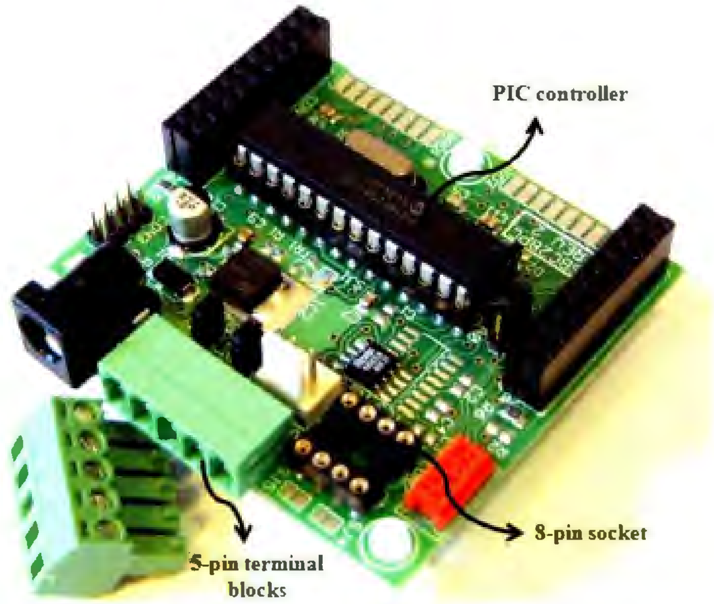

Figure 2.12: The PIC microcontroller based CAN to RS485 converter SBC28PC-IR4 (Modtronix Engineering)

receives the inquiry command from the main CAN-bus, package the required information into specific CAN messages and send them to the inquiry node. When the SBC28PC can not communicate with the motor successfully, the responding CAN messages will be changed to indicate the failure of the motors.

2.2.4.6 Propulsion System

Two Torqeedo Inc. electric outboard motors have been chosen as the propulsion system for the developed ASC. The Torqeedo Inc. Travel 801 motors are lightweight at only 11.57 kg including the weight of the integrated battery (3.5 kg). The maximum input power is 800 W with the supply voltage of 29.6 V, and the corresponding propulsive power is 350 W.

There is a thruster controller built into the enclosure of the Torqeedo thruster which receives the commands from the tiller to control the motor speed, power and the direction. A PID control algorithm in this thruster controller guarantees the propeller rotates at the defined rotational speed. The two electric motors' thruster controllers can be interfaced using the RS485 serial interface, and by issuing different commands, differential thrust can be acquired.

Since the tiller is not installed in the ASC system, a motor controller module that substitutes the role of the tiller has been developed (Figure 2.5). This controller module can issue the control commands for the thrusters as well as log the responding information from the motors. As discussed in the previous section, this module can also be regarded as the protocol converter, because it connects the Torqeedo motors (RS485) to the main communication system (CAN).

A message example used for communication with the Torqeedo motor under the RS485 protocol is shown in Table 2.12. As shown in the table, the information including the message source and destination are included in the first two bytes of the transmitted message. In addition to that, the most important information including the propeller rotational speed, direction and power information are also provided. Upon reception of the motor control command, the motor will respond with its confirmation message, and the corresponding motor specification is configured.

2.2.5 CAN Nodes Development

The details of how different CAN nodes (Figure 2.4 and 2.5) are built using the introduced electrical components are provided.

2.2.5.1 Controller CAN Node Development

The controller CAN node developed for the ASC acts as the command distributor as well as a system information gatherer. This CAN node is designed to receive the commands from the dock-side computer through the wireless modem, and according to

Table 2.12: The messages used to control the Torqeedo motors under the RS485 serial connection

Field definition	Example	Description
Destination address	0x80	Propeller address
Source address	0x10	Tiller address
PCB	0x01	Protocol control byte
INS	0x10	Instruction (Set command)
ID1	0x00	Parameter ID higher byte
ID0	0x12	Parameter ID lower byte
LEN	0x04	Data length
Data1	0x01	Rotational speed higher byte
Data2	0x00	Rotational speed lower byte
Data3	0x01	Direction
Data4	0x32	Power 0 to 100% =0x00 to 0x64
CHK1	-	Checksum higher byte
CHK0	-	Checksum lower byte

the commands, package the CAN messages for inquiry about specific ASC information from different CAN nodes, or send the desired motor configuration commands.

As discussed in the electrical components section, the XTend 900 MHz wireless modem is included to support the wireless communication. The CAN transceiver chip MCP2551 is used to perform the voltage level conversion for the CAN messages. The PB200 WS is also integrated in this CAN node. A 12 V to 5 V DC-DC voltage converter is also used. Figure 2.13 shows the electrical system schematic. As shown, the XTend wireless modem connects to the mbed^{TM} microcontroller through a MAX232 logic level converter, and two CAN driver chips are included. The mbed^{TM} microcontroller has two CAN interfaces. One interface was set to work under the CAN 2.0A standard, and it was used to build the main communication network. The second was set to work under the CAN 2.0B standard, and due to the compliance of the NMEA 2000 with CAN 2.0B, the second CAN interface was used for the weather station NMEA 2000 communication. To protect the WS, a 3 A fuse is also included.

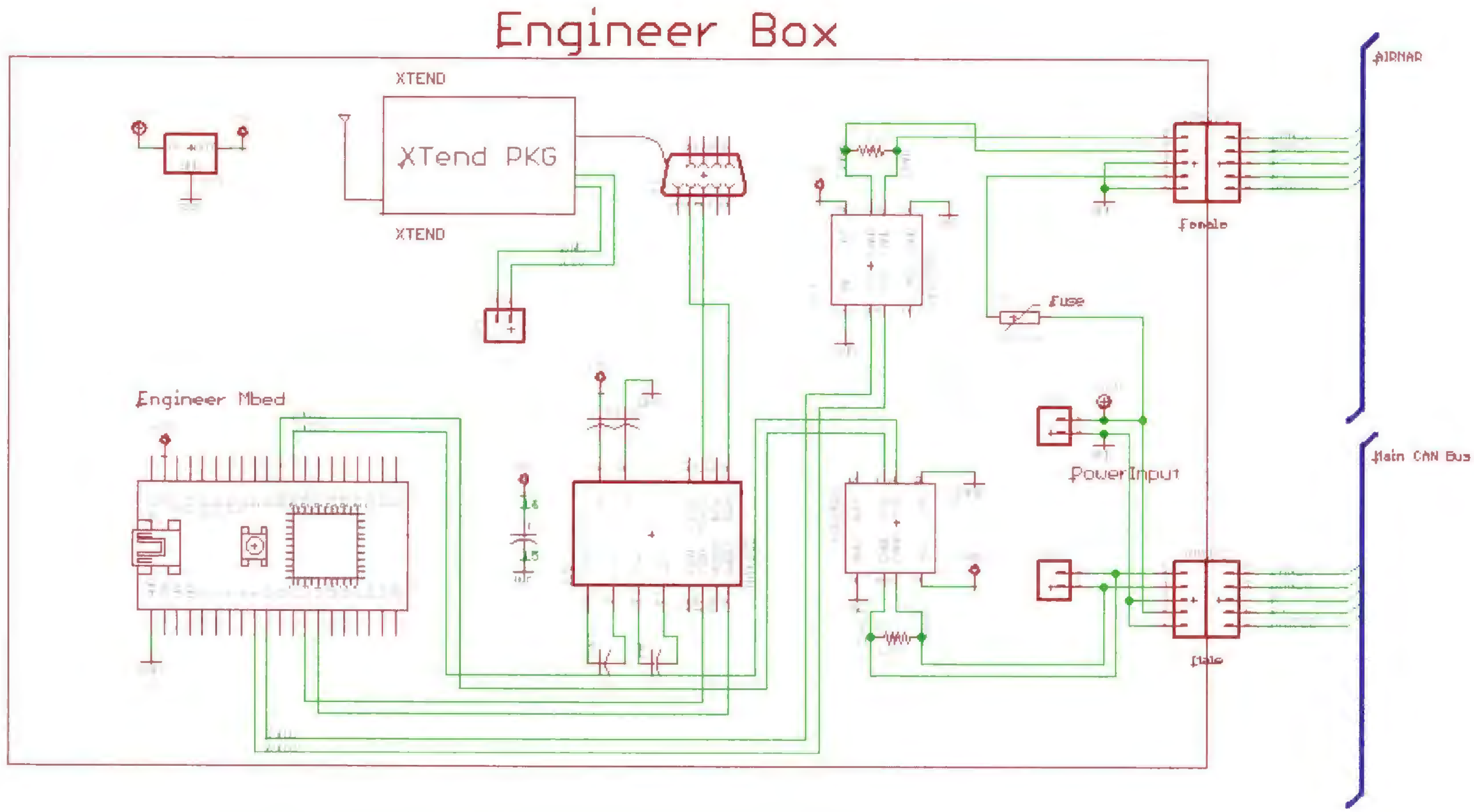

Figure 2.13: The controller CAN node schematic

Figure 2.14: The controller CAN node final realization

Figure 2.14 shows the final realization of the controller CAN node. All the components are soldered onto the prototype board. To make sure the implemented electrical components are water resistant, they are sealed into an aluminum alloy waterproof enclosure that complies with the IP67 standard, and all the cables and connectors implement the same waterproof standard.

2.2.5.2 Navigation CAN Node Development

The navigation CAN node implements an integration of the GPS module and the AHRS module with the microcontroller. The reason to have this CAN node is to automatically log the GPS, heading, acceleration and angular rate information, and after the reception of the request command from the main CAN network, this node will package the corresponding information and send it back to the CAN node that starts the request. Another reason to have this node is to have the sensor data fusion algorithm implemented on-board, and by fusing the information from the GPS and AHRS, a better estimation of location and orientation information can be derived.

Figure 2.15 shows the planned schematic for the navigation CAN node. As shown in the figure, the AHRS is connected with the core processor through the MAX232 logic level converter, and the Grove-GPS is directly interfaced with the mbed^{TM} microcontroller. The connected external antenna is extended to the outside of the navigation box. To connect this CAN node to the main CAN network, the MCP2551 CAN driver chip is used.

According to the planned schematics, the final realization on the prototype board has been completed as shown in Figure 2.16. This CAN node features some similar characteristics as the controller CAN node, such as it also integrates the DC-DC voltage converter for connection to the main CAN-bus, and this CAN node implements the same CAN driver chip MCP2551 for voltage level conversion.

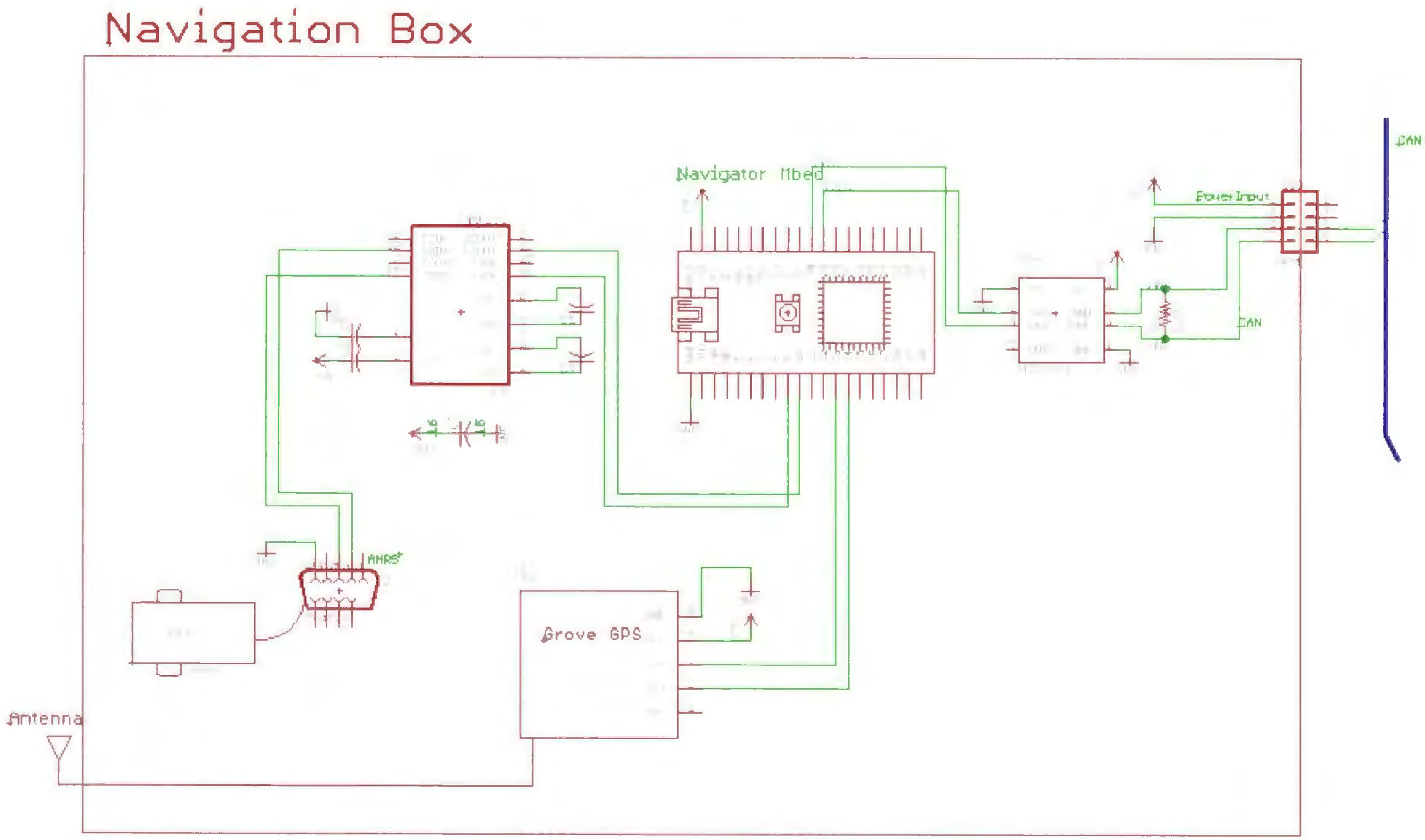

Figure 2.15: The navigation CAN node schematic

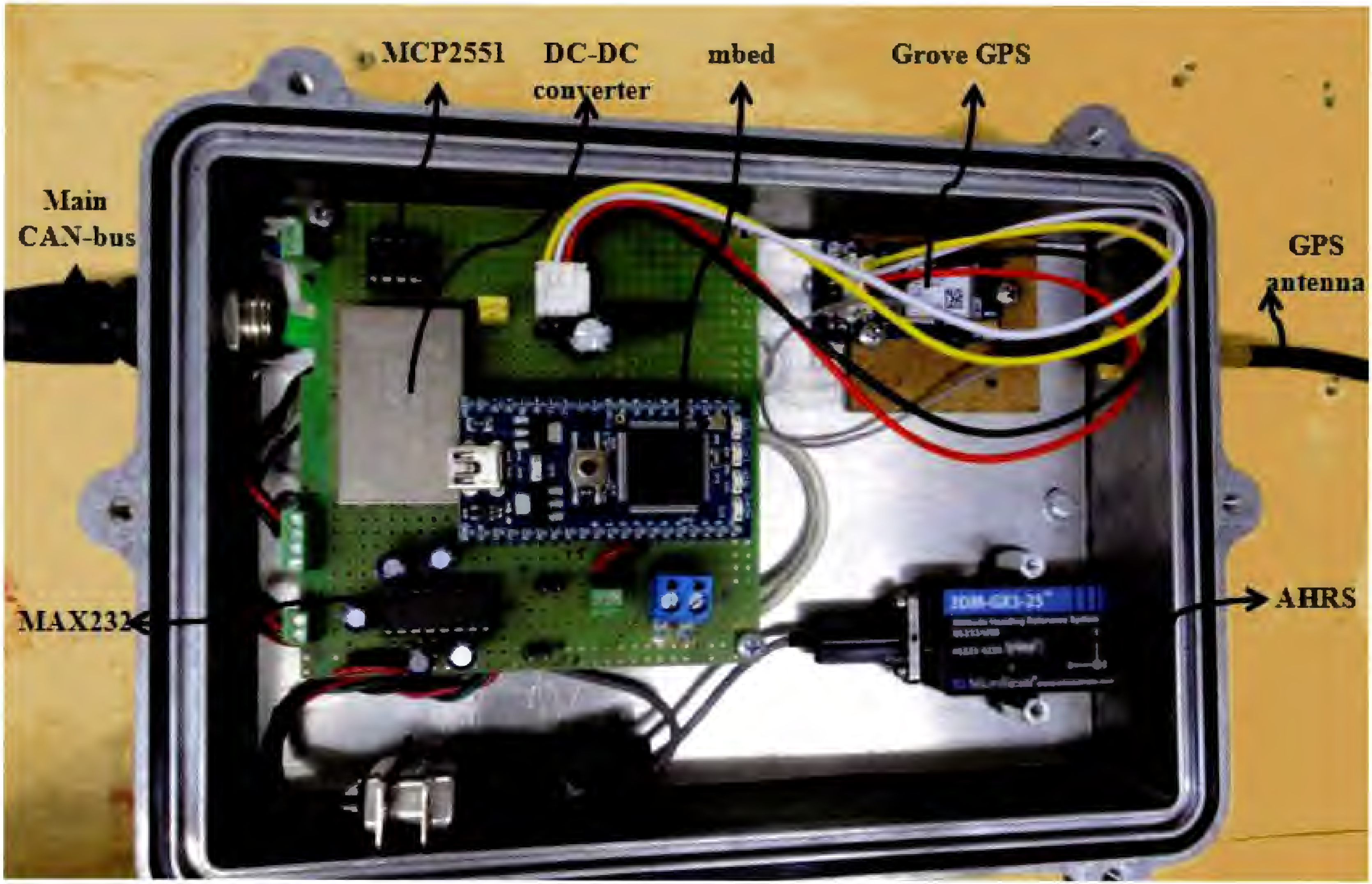

Figure 2.16: The navigation CAN node final realization

2.2.5.3 Motor Controller CAN Nodes Development

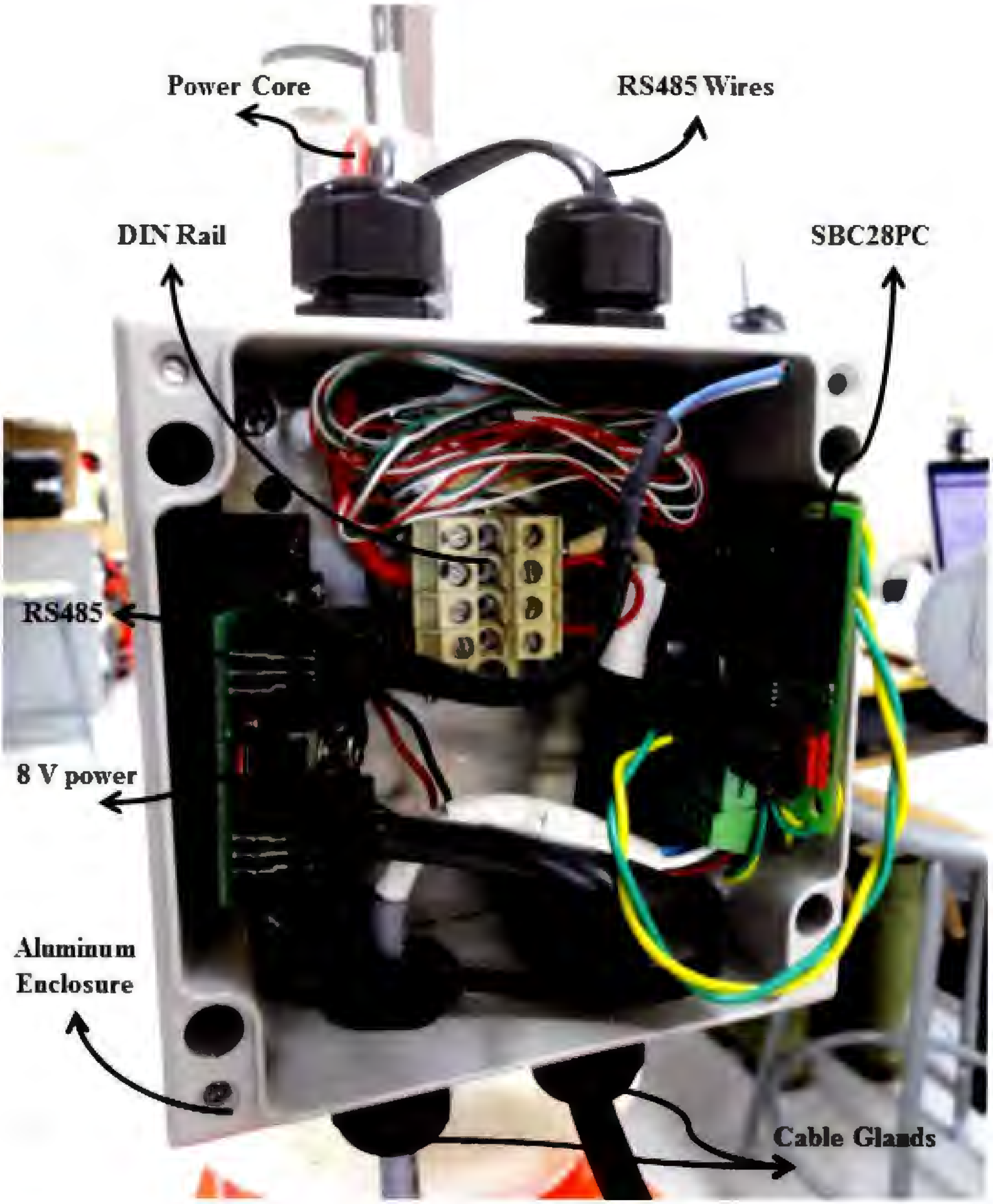

Figure 2.17: Motor controller CAN node final realization

Since the communication with the Torqeedo propellers is based on the RS485 serial interface, the CAN messages have to be converted to the RS485 format. The main task of the CAN nodes will be conversion of the protocol between the RS485 and the CAN standard, and deliver the power to the propulsion system as well as the 8V voltage required for RS485 message transmission.

It has been decided that a couple of components have to be included in this CAN node design. As shown in Figure 2.17, the SBC28PC board has been used as the main microcontroller for protocol conversion. The voltage converter board that comes with the Torqeedo propellers is used for generating the proper voltage for RS485 communication. The DIN rails are used to connect the power lines to the propellers. All the components are enclosed inside a rugged aluminum alloy box, and all the cables running out are sealed with the specially chosen cable glands. The proposed design is validated to provide the proper functionality and the satisfactory waterproofness.

2.3 Program and Software Development

In order to build the decentralized communication system for the ASC, each connected CAN node has to be developed with its own program. The details of these separately developed programs are introduced. To make it possible to display the ASC on-board sensor information on the dock-side computer as well as issue the supervisory command, a Matlab based Graphic User Interface (GUI) that runs on the dock-side computer is designed.

2.3.1 CAN Nodes Program Development

There are a total of four CAN nodes developed to work on the main CAN network. Among these four nodes, the controller CAN node and navigation CAN node are programmed with the ARM processor, while the motor controller CAN nodes are programmed with the PIC microcontroller.

Table 2.13: CAN data frame identifiers allocation

ID	Description	Message Source
1	Time Reference Message (TRM)	Node 2
2	Error Message from Left Motor	Node 4
3	Error Message from Right Motor	Node 3
4	Left Motor Set	Node 1
5	Right Motor Set	Node 1
6	Left Motor Speed, Direction and Power	Node 4
7	Right Motor Speed, Direction and Power	Node 3
16	GPS Data-Longitude and Latitude	Node 2
17	GPS Data-Speed Over Ground and Course Over Ground	Node 2
18	Acceleration in X and Y Axes	Node 2
19	Acceleration in Z Axis and Angular Rate Around X Axis	Node 2
20	Angular Rate Around Y and Z Axes	Node 2
21	Magnetometer in X and Y Axes	Node 2
22	Magnetometer in Z axis and Rotation Matrix $M_{1,1}$	Node 2
23	Rotation Matrix $M_{1,2} - M_{3,3}$	Node 2

Table 2.14: CAN remote frame identifiers allocation

ID	Description
4	Left motor status inquiry
7	Right motor status inquiry
16	GPS data-longitude and latitude
17	GPS data-longitude, latitude, speed and course over ground
18	AHRS data-acceleration in x, y and z axes
19	AHRS data-angular rate around x, y and z axes
20	AHRS data-all acceleration and angular rate information
21	GPS data, acceleration and angular rate information
22	AHRS data-magnetometer in x, y and z axis
23	AHRS data-rotational matrix

To guarantee the whole system works properly, an allocation of the CAN message identifiers is performed. As shown in Table 2.13, the highest priority ID is allocated to the TRM since it triggers the whole system to work under the same time step and is needed to be transmitted even if a transmission conflict occurs. The two motors are essential for propulsion, so the motor control and status information is assigned the next higher priority level. Following that is the information about GPS position, speed, heading, acceleration, and other sensor data. Since some of the CAN messages from Table 2.13 are requested using the remote frame, an ID allocation of the remote frame CAN messages is also provided and shown in Table 2.14.

2.3.1.1 The Controller CAN Node Program Development

The controller CAN node is designed to complete the following tasks.

- Send the request CAN message to the navigation CAN node to get the vehicle related navigation information
- Acquire the motor information and send configuration commands to change the speed, direction and power
- Obtain the information from the Airmar weather station and log the data
- Communicate with the dock-side computer through the wireless connection and transmit sensor data and receive supervisory commands.

In order to show a clear picture of the working process of the controller CAN node, the program flow chart is provided in Figure 2.18 and the main function C++ codes are provided in Appendix A.1.

In the CAN interface initialization part, two CAN interfaces are defined. One CAN interface is configured to work in the standard mode with the communication baud rate of 1 Mbps, and the other is configured to work in the extended mode with the baud rate of 250 kbps to communicate with the WS. After that, the controller CAN node waits for the TRM from the navigation CAN node, and after successful reception of the TRM, it will package a TRM wireless message to be transmitted

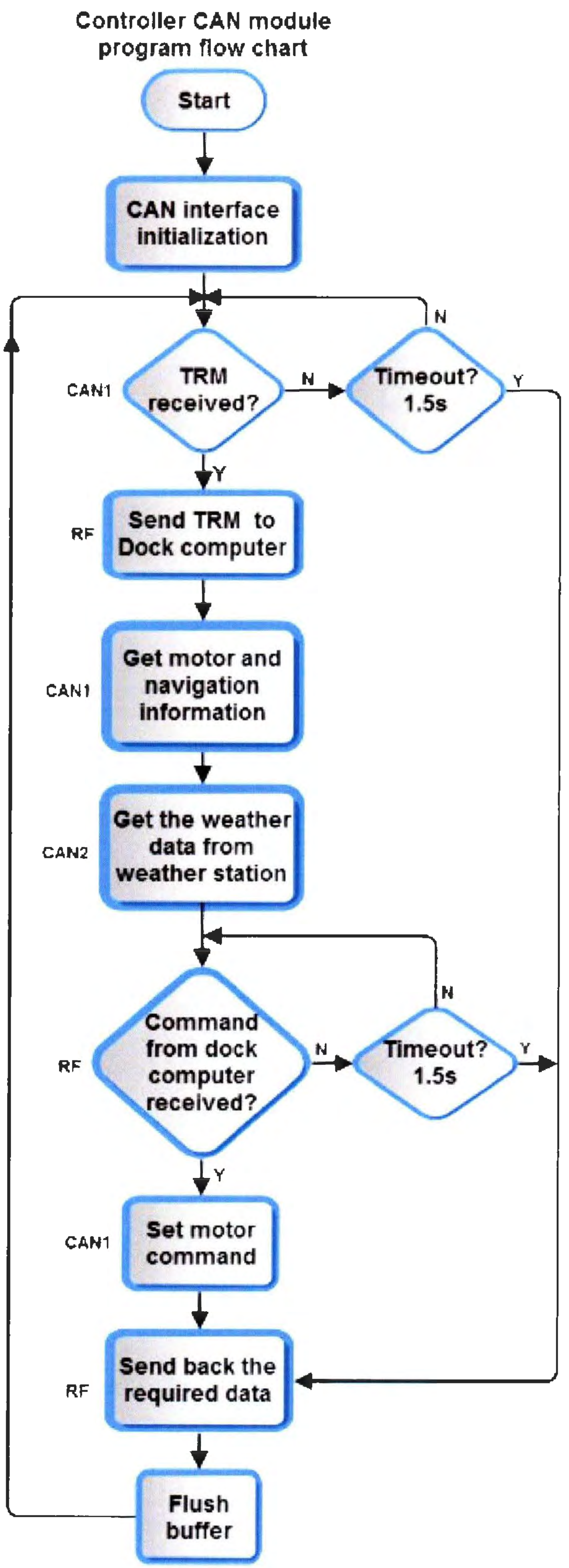

Figure 2.18: The controller CAN node program flow chart

to the dock-side computer for time synchronization purposes. Following that, the motor and navigation information are requested from the other CAN nodes by the controller CAN node, and then the supervisory commands will be received from the dock-side computer. Based on the supervisory commands, the motor status will be reconfigured, and required sensor data will be send back to the dock-side computer. As shown in the flow chart, to make sure this CAN node is not locked with any wait function, two 1.5 second timeout functions are attached.

2.3.1.2 The Navigation CAN Node Program Development

The navigation CAN node is responsible for collecting data from the GPS and AHRS modules. Normally a sensor fusion algorithm (Kalman filter) is implemented on the microprocessor to fuse the information for better estimation of the vehicle status; however, in this design, the concentration is on the construction of the CAN-bus based communication and control system structure, so no navigation algorithm is implemented yet.

The flow chart in Figure 2.19 shows a clear working process for this CAN node, and the main function C++ codes are provided in Appendix A.2. The mbed^{TM} microcontroller in the navigation CAN node works under the trigger from the GPS GPRMC message which has been configured to be updated every 1 second. After the mbed^{TM} microcontroller starts inquiring about the RMC message, it keeps waiting until there is a response, and then it packages the TRM CAN message using the UTC time information and sends it onto the main CAN network to indicate the beginning of this time period. After this, the navigation CAN node continues to get the information from the AHRS and packages it with the GPS data for further navigation algorithm usage. The navigation information is requested in the interrupt routine.

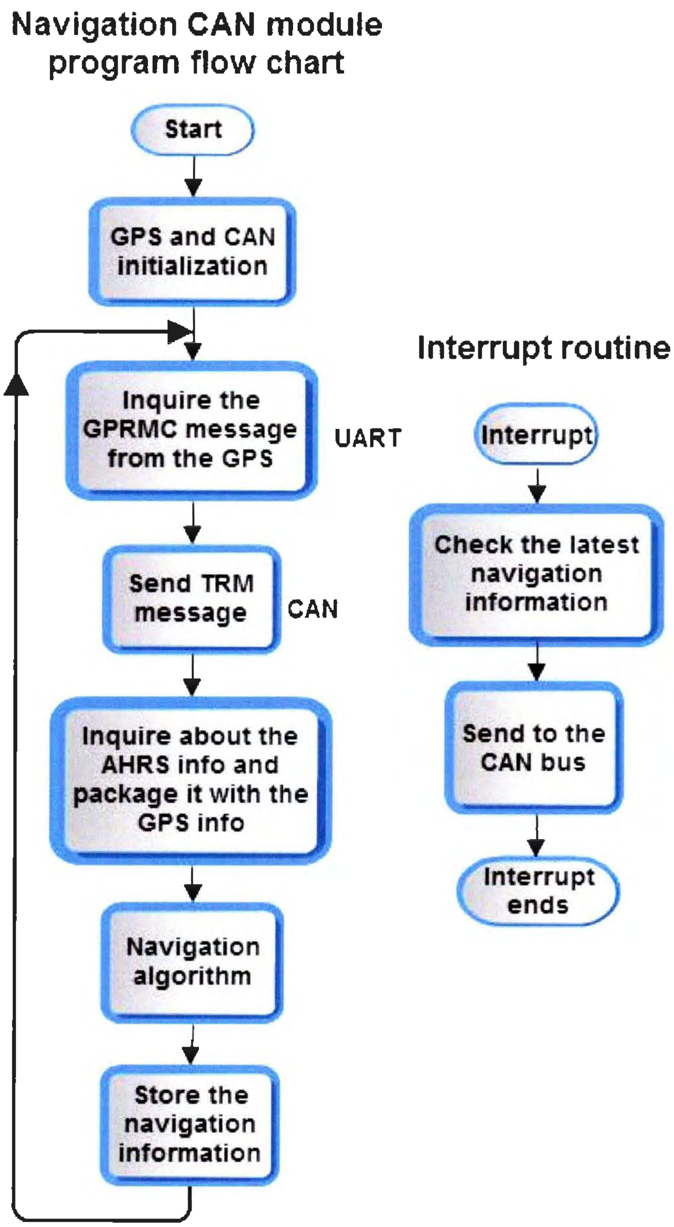

Figure 2.19: The navigation CAN node program flow chart

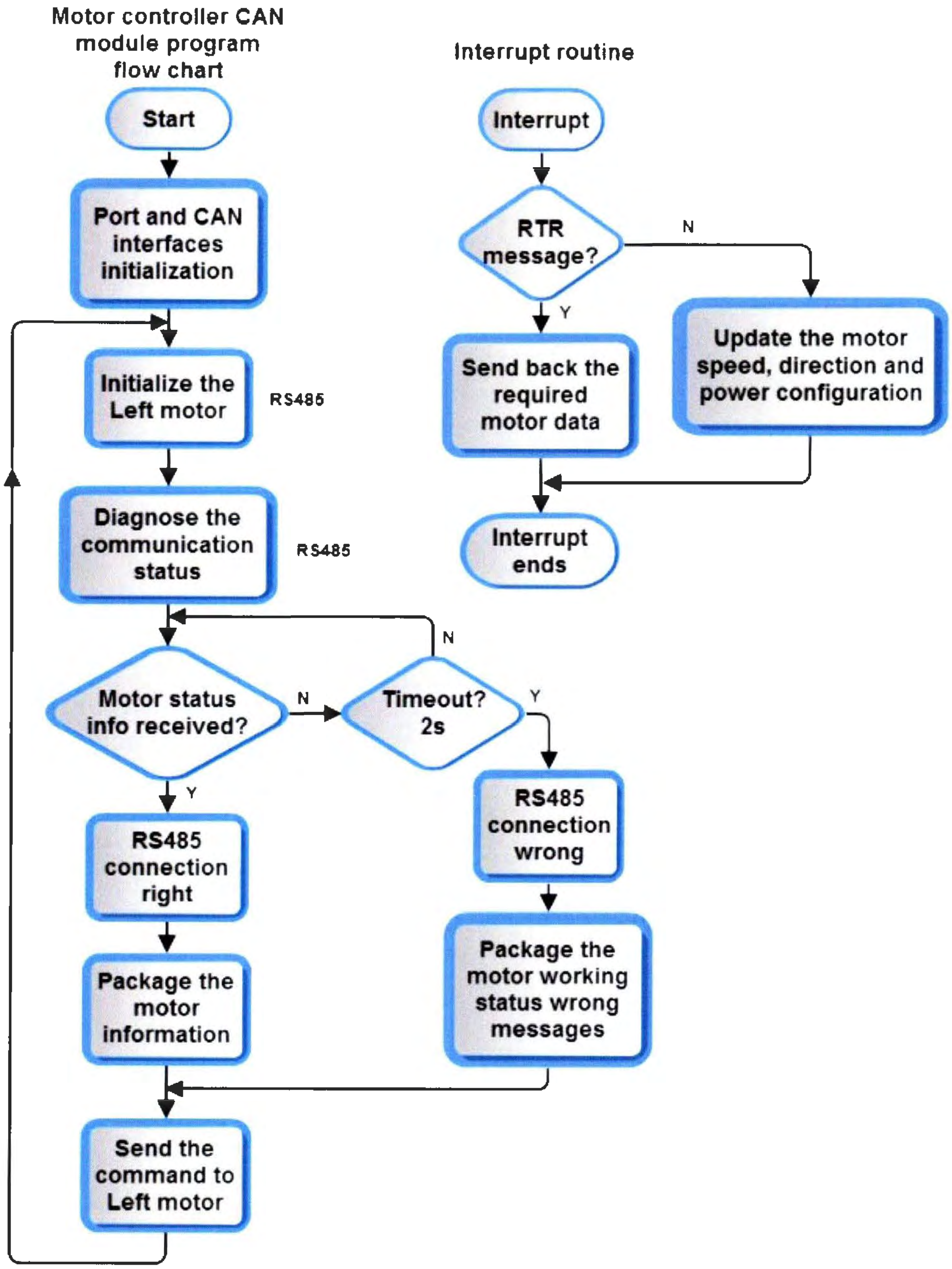

Figure 2.20: The motor controller CAN node program flow chart

2.3.1.3 The Motor Controller CAN Node Program Development

Since communication with the Torqeedo motors is restricted to RS485 communication, the main task for the motor controller CAN node is to convert the messages between the CAN protocol and the RS485 format. In addition to the protocol conversion, this CAN node is designed to be capable of diagnosing the motor working status and logging the required motor information for transmission to other inquiry CAN nodes. Figure 2.20 shows the flow chart of the developed program for the left motor controller, and the C code is provided in Appendix A.3. As shown, the motor configuration keeps updating, and after setting the motor each time, the motor will respond with the confirmation message. Taking advantage of this feature, the motor core microcontroller can decide the RS485 communication status. The motor speed modification is completed by using the interrupt routine, which can guarantee that the motor configuration is updated with the minimum delay. The preprocessors are used for conditional compilation. For example, if the user wants to compile the code for the right side motor controller (refer to Appendix A.3 Page 130 Line 15), it is only necessary to set the global variable "L_or_R" to be 0, or define it to be 1 for the left motor.

2.3.1.4 System Time Synchronization and Evaluation

As introduced in the Time-Triggered CAN (TTCAN) section, the TRM based system time synchronization method is implemented in the development of the main communication and control system. The TRM is a standard CAN message that includes UTC time information from the navigation CAN node. When the TRM is accepted by the other CAN nodes on the main CAN network, they will know the start of this data transmission cycle begins, and all the connected CAN modules will be synchronized to the same time signal.

In order to show the achieved characteristics of the TRM based system time synchronization method, an evaluation test has been performed using the DPO4000 series digital phosphor oscilloscope. The DPO4000 series oscilloscope is capable of displaying the CAN-bus information, and with its built-in functions, CAN messages can be identified.

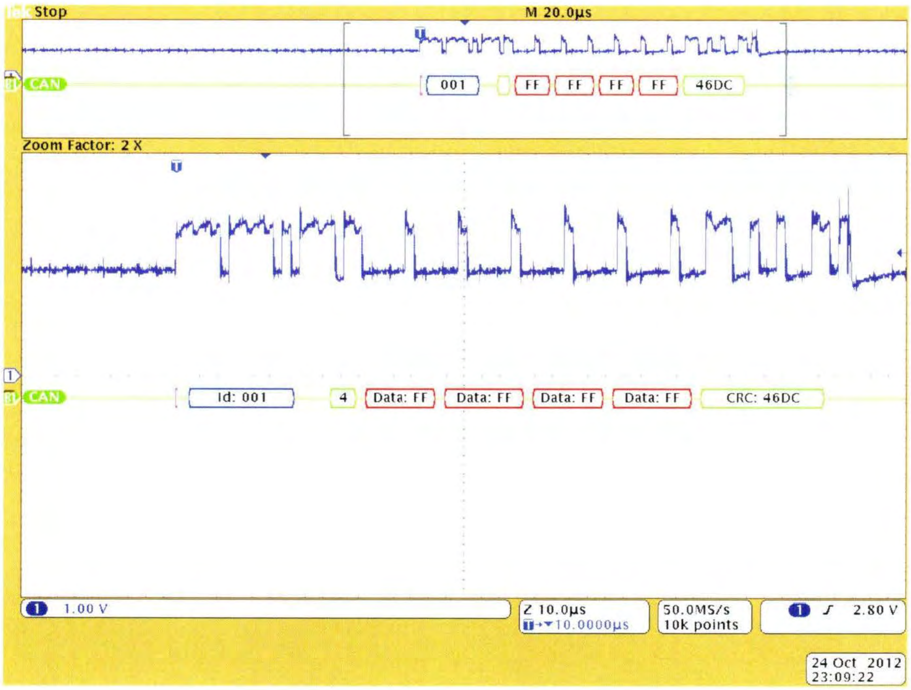

Figure 2.21: The TRM message organization

In the evaluation test, the DPO4034 is connected to the CANH line and the ground to display the transmitted CAN messages within the designed CAN network. The first step of the test is to identify the TRM message. To do this, the main communication network is assigned to work only with the TRM message. Figure 2.21 shows the captured TRM message from this test, and the square waves indicates the transmitted TRM. The decoding of the square waves is automatically done by the oscilloscope, and the interpreted hex number is shown underneath each transmitted data byte. Inside these hex numbers, the first number, 001, is TRM ID number, and following that are the data length, which is 4 in this case. Since the ASC system was tested inside the building, the GPS can not get fixed data, so the default four data bytes, FF FF FF FF, were transmitted instead of the UTC time data. The Cyclic Redundancy Check (CRC) number is 46DC. The data transmission rate is 1 Mbps, and the time for the transmission of the TRM is 84 μs. Another test is performed to validate that the TRM message is transmitted exactly each second, and the result is shown in Figure 2.22.

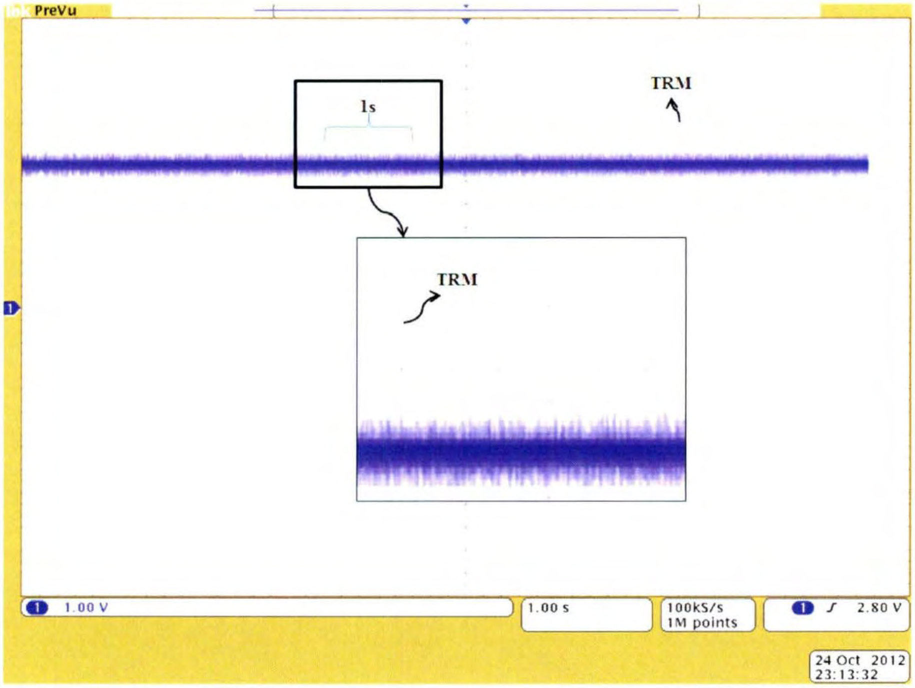

Figure 2.22: The time interval between TRM messages

For the next step of the test, the main communication system is configured to work normally with required information transmitted on the CAN-bus. In this normal working mode, navigation information including GPS location, speed and course over ground, acceleration in three axes, angular rate, magnetic field, and the motor information including direction, speed and power are transmitted on the CAN network. Figure 2.23 shows one capture of the transmitted sensor and motor data on the CAN-bus. In order to use the CAN-bus more efficiently, the data are requested and transmitted consecutively, and it can be seen from Figure 2.23 that the total time used for the transmission of all these messages takes up to 1,200 μs (three grid squares and each grid square is 400 μs as shown in Figure 2.23).

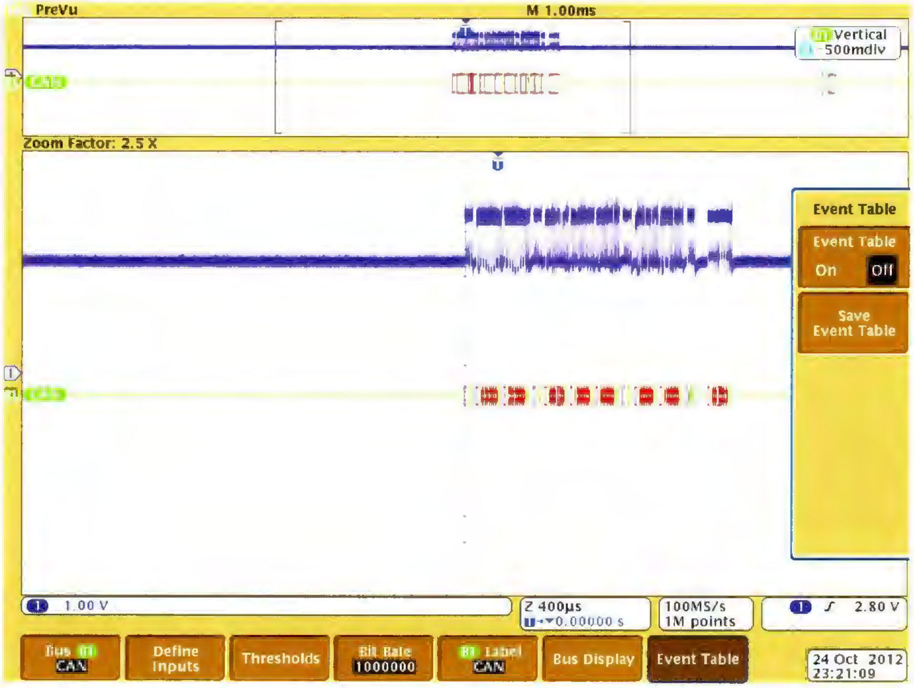

Figure 2.23: Captured sensor and motor CAN messages

Using the DPO4034 event table function, the transmitted data are extracted and shown in Table 2.15. The sensor and motor data are requested by issuing the remote frame CAN messages. For example, after the remote frame message with the identifier 0x11 is sent onto the CAN-bus, the data frames with the identifier 0x10 (GPS longitude and latitude) and 0x11 (speed and course) will be sent to the inquiry CAN node.

The CAN-bus load for each second is calculated as shown in Equation 2.1. It can be concluded that, although all time critical CAN messages are transmitted, there is still about 99% space left for other types of messages transmission.

$$Load = (1200\mu s + 84\mu s)/(1 * 10^6\mu s) = 0.128\% \qquad (2.1)$$

Table 2.15: Captured sensor and motor CAN messages list

Time	Identifier	DLC	Data	CRC
-1.49E-04	0x11	8	Remote Frame	5C2E
-9.60E-05	0x10	8	00 00 00 00 00 00 00 00	6072
3.10E-05	0x11	8	00 00 00 00 00 00 00 00	6B69
1.61E-04	0x14	8	Remote Frame	3.30E+09
2.15E-04	0x12	8	BC 50 9B F7 BC 8F DF FD	1C6D
3.31E-04	0x13	8	BF 80 76 64 BB 91 CC 70	6.30E+03
4.46E-04	0x14	8	BA E2 59 AA 3C 36 AF A9	2B14
5.66E-04	0x16	8	Remote Frame	77CE
6.21E-04	0x15	8	C1 94 A3 CE 43 4D 7F 6C	7AD4
7.35E-04	0x16	8	41 E4 02 84 BD 96 9D 1B	15C8
8.56E-04	0x04	8	Remote Frame	4A17
9.48E-04	0x02	8	20 E4 02 52 00 00 00 00	513A
4.02E-03	0x07	8	Remote Frame	2C22

2.3.2 Matlab Based GUI Software Design

Through the wireless communication link, the vehicle supervisory commands and vital ASC operation status information can be exchanged between the ASC and the dock-side computer. This system is important during the ASC testing, which will be introduced in Chapter 4. Using the wireless system, all ASC sensor and motor data are synchronized to the dock-side computer for on-line analysis, and it is also possible to send the commands to guide the ASC to work.

In order to use this wireless link to transmit the ASC status information and control commands, a well planned program schedule is required. The flow charts in Figure 2.24 show the software running on the dock-side computer and how it coordinates with the controller CAN node program.

In Figure 2.24, the TRM will first be transmitted to the dock-side computer using the wireless link, and then based on the user's configuration, a supervisory command will be sent back to the ASC. This command is interpreted into the concrete operation inside the ASC, such as the navigation data are requested and motor speed is changed. After this operation, the desired information is sent back to dock-side computer to be displayed or logged.

A Matlab based GUI is developed to work on the dock-side computer as a control terminal for the ASC. Figure 2.25 shows the final realized Matlab GUI. The sensor and motor data from the ASC can be shown on the GUI in quasi-realtime, and it is convenient to control the ASC two motors by using the motor control function. A Bluetooth hand controller is integrated for more intuitive control of ASC, and the GUI can directly log the ASC location into the Google Earth software.

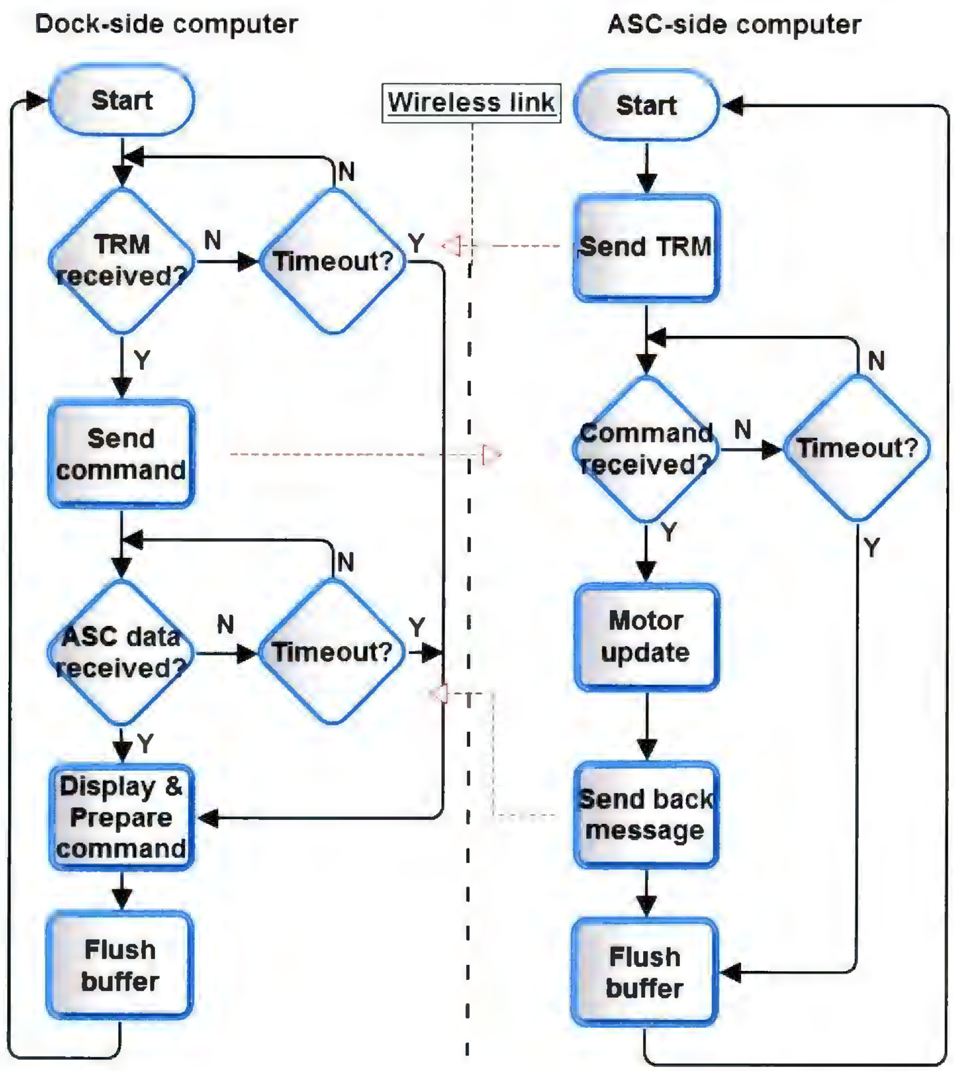

Figure 2.24: Cooperation of the dock-side software with the controller CAN node program

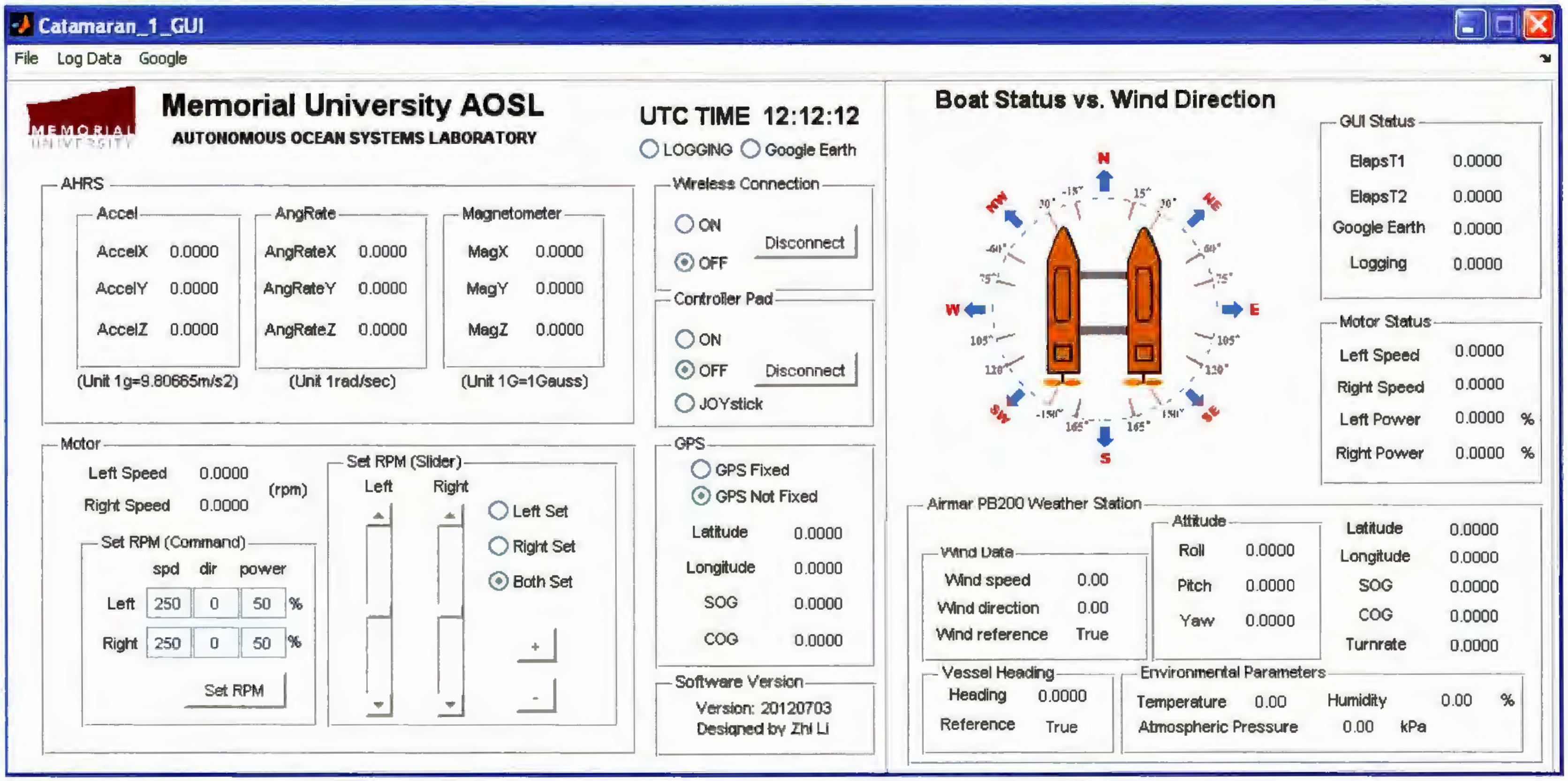

Figure 2.25: Matlab based GUI for the ASC system

Chapter 3

Mathematical Model for the Autonomous Surface Craft

3.1 Nonlinear Model for the ASC

The notation used for describing the general motion of the developed ASC is provided in Figure 3.1. The origin of the body-fixed frame (point o) is chosen to be inside the ASC's xz plane (the designed ASC has xz plane symmetry), and then the body fixed coordinate system is defined as:

- ox axis is directed from aft to fore
- oy is directed to starboard
- oz is directed from top to bottom.

In addition to that, the $o_E x_E y_E z_E$ defines a coordinate frame that is fixed on the Earth, and since the Earth rotation will not affect the ASC motion, this frame can be regarded as an inertial frame. Taking advantage of these two frames, the vehicle status information including velocity and angular rate expressed in the body-fixed

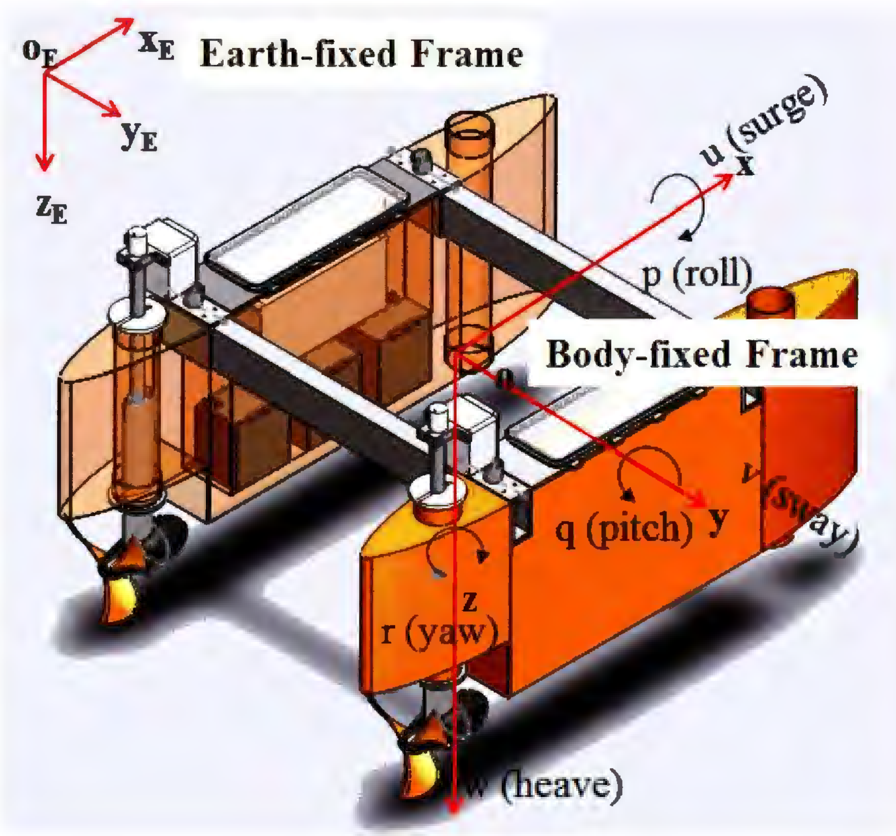

Figure 3.1: Notation for ASC

frame can be converted to the inertial frame. A summary of these terms and their definition from the Society of Naval Architects and Marine Engineers (SNAME) is included in Table 3.1 [30].

By using these terms, the ASC motion can be described using the six degrees of freedom (DOF) motion equations (refer to [30] for details). However, since the ASC motions in heave, roll and pitch is small in most cases, a 3 DOF model that only considers the ASC movement in the horizontal plane (surge, sway and yaw) is provided in this model development process.

Before this 3 DOF model is generated, the following vectors are defined according to the SNAME notation.

Table 3.1: SNAME notations

	Forces & Moments	Linear Velocity & Angular Velocity	Position & Euler Angle
Motion along ox axis (surge)	X	u	x
Motion along oy axis (sway)	Y	v	y
Motion along oz axis (heave)	Z	w	z
Rotation around ox axis (roll)	K	p	ϕ
Rotation around oy axis (pitch)	M	q	θ
Rotation around oz axis (yaw)	N	r	ψ

- $v = [u\ v\ r]^T$: surge and sway velocity, and yaw angular velocity expressed in body-fixed frame
- $\eta = [x\ y\ \psi]^T$: x and y location and yaw angle expressed in the inertial frame
- $o_G = [x_G\ y_G\ z_G]^T$: vector pointing from the body-fixed frame origin to the center of gravity (CG)

Based on this vector definition, a compact 3 DOF kinematic and dynamic model expression can be achieved as shown in Equation 3.1 [30].

$$\begin{cases} \dot{\eta} = Rv \\ M\dot{v} + C(v)v + Dv = \tau \end{cases} \tag{3.1}$$

In this model, R defines the rotation matrix that converts the speed vector from the body-fixed frame to the inertial frame, and therefore the kinematic model can be rewritten as follows:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ r \end{pmatrix} \tag{3.2}$$

In the dynamic model, M is the mass matrix, and C is the Coriolis and centripetal matrix, D is the damping matrix and τ is external force and torque (for the purpose of this model the restoring forces in heave, roll and pitch are neglected since the ASC motions in heave, roll and pitch are small). A detailed description of these terms are provided as:

- M is a combination of vehicle inertia (M_{RB}) and added mass (M_A) due to the inertia of the surrounding fluid, namely, $M = M_{RB} + M_A$
- C(v) includes the Coriolis and centripetal force contributed from the vehicle itself and the added mass effect, namely, $C(v) = C(v)_{RB} + C(v)_A$
- D, the damping matrix of the vehicle, comes from effects including the radiation-induced potential damping due to the energy carried away by generated surface waves, skin friction, wave drift damping and damping due to vortex shedding
- τ consists of environmental forces (currents, waves and wind) and propulsion and rudder forces

To further simplify the terms M, C(v) and D inside the dynamic model of Equation 3.1, the following conditions are assumed [32]:

- Motion in heave, roll and pitch is neglected
- Environmental forces due to wind, currents and waves are excluded

- The ship has homogeneous mass distribution and xz-plane symmetry
- Center of gravity (CG) and center of buoyancy (CB) are located vertically on the same z-axis
- Assume the inertia added mass and damping matrices are diagonal

As a result of these assumptions, the simplified dynamic model terms M, C(v) and D are given in Equations 3.3 to 3.5.

$$M = \begin{pmatrix} m - X_{\dot{u}} & 0 & 0 \\ 0 & m - Y_{\dot{v}} & 0 \\ 0 & 0 & I_z - N_{\dot{r}} \end{pmatrix} \tag{3.3}$$

$$C(v) = \begin{pmatrix} 0 & 0 & (Y_{\dot{v}} - m)v \\ 0 & 0 & (m - X_{\dot{u}})u \\ (m - Y_{\dot{v}})v & (X_{\dot{u}} - m)u & 0 \end{pmatrix} \tag{3.4}$$

$$D = \begin{pmatrix} -X_u & 0 & 0 \\ 0 & -Y_v & 0 \\ 0 & 0 & -N_r \end{pmatrix} \tag{3.5}$$

A redefinition of the coefficients in Equations 3.3 to 3.5 is shown in Equation 3.6 for a compact formula expression. Using the newly defined coefficients in Equation 3.6, a compact model describing the dynamic motion of the ASC is shown in Equation 3.7. In Equation 3.7, u_1 and u_2 stand for the applied external forces along the surge and sway direction, and u_3 defines the steering torques around the z axis which is given by the product of the thrust produced by each propeller and the distance that each propeller is offset from the longitudinal centreline (0.5 m as stated in Table 2.1). The state variables $[u\ v\ r]^T$ follow the SNAME definition, and among the constants, m_{ii} $(i = 1, 2, 3)$ are determined by ASC inertia and added mass effects, and d_{ii} $(i = 1, 2, 3)$ are determined by the hydrodynamic effects.

$$\begin{cases} m_{11} = m - X_{\dot{u}} \\ m_{22} = m - Y_{\dot{v}} \\ m_{33} = I_z - N_{\dot{r}} \\ d_{11} = -X_u \\ d_{22} = -Y_v \\ d_{33} = -N_r \end{cases} \tag{3.6}$$

$$\begin{cases} \dot{u} = \frac{m_{22}}{m_{11}} \cdot v \cdot r - \frac{d_{11}}{m_{11}} \cdot u + \frac{1}{m_{11}} \cdot u_1 \\ \dot{v} = -\frac{m_{11}}{m_{22}} \cdot u \cdot r - \frac{d_{22}}{m_{22}} \cdot v + \frac{1}{m_{22}} \cdot u_2 \\ \dot{r} = \frac{m_{11} - m_{22}}{m_{33}} \cdot u \cdot v - \frac{d_{33}}{m_{33}} \cdot r + \frac{1}{m_{33}} \cdot u_3 \end{cases} \tag{3.7}$$

Equations 3.2 and 3.7 together are the simplified ASC 3 DOF nonlinear model. However, since in this ASC design no rudder is installed on the vehicle, there is no direct control of the sway motion. Therefore, a proper model that can describe the kinematic and dynamic motion of the designed ASC will neglect the sway control input u_2. The complete 3 DOF model for the designed ASC is summarized in Equation 3.8.

$$\begin{cases} \dot{u} = \frac{m_{22}}{m_{11}} \cdot v \cdot r - \frac{d_{11}}{m_{11}} \cdot u + \frac{1}{m_{11}} \cdot u_1 \\ \dot{v} = -\frac{m_{11}}{m_{22}} \cdot u \cdot r - \frac{d_{22}}{m_{22}} \cdot v \\ \dot{r} = \frac{m_{11}-m_{22}}{m_{33}} \cdot u \cdot v - \frac{d_{33}}{m_{33}} \cdot r + \frac{1}{m_{33}} \cdot u_3 \\ \dot{x} = ucos(\psi) - vsin(\psi) \\ \dot{y} = usin(\psi) + vcos(\psi) \\ \dot{\psi} = r \end{cases} \tag{3.8}$$

As shown in Equation 3.8, this generated model can be divided into two groups. The first three equations consist of the first group which describes the dynamic motion of the ASC. By using this model, when a proper system input is applied, the ASC dynamic and steady state motion can be calculated.

Using the first group of equations, ASC status vector $v = [u\ v\ r]^T$ can be generated. By implementing the last three equations, the ASC position and orientation can be expressed in the Earth-fixed frame. The second group of equations can be regarded as the coordinates transformation matrix. It is reasonable to design a controller only for the dynamic model, and then use the second group of equations for the coordinate transformation.

To design the linear control algorithm for the designed ASC, a linear ASC model has to be used. Subsequently, two methods for generating the linear dynamic model are introduced.

3.2 Linear Model for the ASC

3.2.1 Linear Model Generation using Taylor Series Expansion

Based on the nonlinear model in Equation 3.8, the Taylor series expansion is used to generate the corresponding linear model. First, an equilibrium point for the nonlinear model has to be defined. This equilibrium point is quite important, because the linearized model is only valid within a small range of this point.

The equilibrium point has been defined as ASC moving in a straight line with a constant forward speed. Under this assumption, the vehicle surge velocity will be constant value $u_0 = u_0^*$, while the sway velocity (v_0) and yaw angular velocity (r_0) will be zero. The propulsion force from two propellers will be equal and constant value $u_{10} = u_1^*$, and the steering torque (u_{30}) is zero.

$$e_0 = [u_0, v_0, r_0, u_{10}, u_{30}]^T = [u_0^*, 0, 0, u_1^*, 0]^T \tag{3.9}$$

Then the dynamic model can be linearized around e_0 by using the Taylor series expansion form as stated in Equation 3.10. In this equation, a_1 to a_d define the equilibrium points.

$$f(x_1, ...x_d) = \sum_{n_1=0}^{\infty} \sum_{n_2=0}^{\infty} ... \sum_{n_d=0}^{\infty} \frac{(x_1 - a_1)^{n_1}...(x_d - a_d)^{n_d}}{n_1!...n_d!} \left(\frac{\partial^{n_1+...+n_d} f}{\partial x_1^{n_1}...\partial x_d^{n_d}}\right)(a_1...a_d) \tag{3.10}$$

The following steps are used to obtain the final linearized model.

$$\begin{cases} \dot{u} = f_1(u, v, r, u_1)|_{e0} = (\frac{m_{22}}{m_{11}} \cdot v \cdot r - \frac{d_{11}}{m_{11}} \cdot u + \frac{1}{m_{11}} \cdot u_1)|_{e0} \\ \dot{v} = f_2(u, v, r)|_{e0} = (-\frac{m_{11}}{m_{22}} \cdot u \cdot r - \frac{d_{22}}{m_{22}} \cdot v)|_{e0} \\ \dot{r} = f_3(u, v, r, u_3)|_{e0} = (\frac{m_{11}-m_{22}}{m_{33}} \cdot u \cdot v - \frac{d_{33}}{m_{33}} \cdot r + \frac{1}{m_{33}} \cdot u_3)|_{e0} \end{cases} \tag{3.11}$$

The partial differentiation of Equation 3.11 is shown in Equation 3.12. To get a linear model, only the first derivatives of Taylor series expansion are kept.

$$\begin{cases} f_1|_{e0} = \frac{\partial f_1}{\partial u} \cdot (u - u_0) + \frac{\partial f_1}{\partial v} \cdot (v - v_0) + \frac{\partial f_1}{\partial r} \cdot (r - r_0) + \frac{\partial f_1}{\partial u_1} \cdot (u_1 - u_{10}) \\ f_2|_{e0} = \frac{\partial f_2}{\partial u} \cdot (u - u_0) + \frac{\partial f_2}{\partial v} \cdot (v - v_0) + \frac{\partial f_2}{\partial r} \cdot (r - r_0) \\ f_3|_{e0} = \frac{\partial f_3}{\partial u} \cdot (u - u_0) + \frac{\partial f_3}{\partial v} \cdot (v - v_0) + \frac{\partial f_3}{\partial r} \cdot (r - r_0) + \frac{\partial f_3}{\partial u_3} \cdot (u_3 - u_{30}) \end{cases} \tag{3.12}$$

Therefore, Equation 3.13 is obtained.

$$\begin{cases} \dot{u} = -\frac{d_{11}}{m_{11}} u + \frac{1}{m_{11}} u_1 \\ \dot{v} = -\frac{d_{22}}{m_{22}} v - \frac{m_{11}}{m_{22}} u_0^* r \\ \dot{r} = \frac{m_{11}-m_{22}}{m_{33}} u_0^* v - \frac{d_{33}}{m_{33}} r + \frac{1}{m_{33}} u_3 \end{cases} \tag{3.13}$$

Finally, the 3 DOF linear model is expressed in state space form.

$$\begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} -\frac{d_{11}}{m_{11}} & 0 & 0 \\ 0 & -\frac{d_{22}}{m_{22}} & -\frac{m_{11}}{m_{22}} u_0^* \\ 0 & \frac{m_{11}-m_{22}}{m_{33}} u_0^* & -\frac{d_{33}}{m_{33}} \end{pmatrix} \begin{pmatrix} u \\ v \\ r \end{pmatrix} + \begin{pmatrix} \frac{1}{m_{11}} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1}{m_{33}} \end{pmatrix} \begin{pmatrix} u_1 \\ 0 \\ u_3 \end{pmatrix} \tag{3.14}$$

Equation 3.14 shows the linear model generated using a Taylor series expansion. In order to implement this model for controller design or ASC system simulation, all the coefficients (m_{ii} and d_{ii} $(i = 1, 2, 3)$) have to be identified using the experiments or simulation. Some of the experiments are time-consuming, and the accuracy of the coefficients depends on the experimental measurements. Therefore, another convenient method for generating the linear model from the nonlinear dynamic equations is introduced.

3.2.2 Linear Model Generation using the System Identification

The system identification (SI) technique is widely used for identification of a relatively complicated system process (e.g. ; chemical process). In the identification process, a well planned input signal is injected into an identified object, and the output signals are recorded. Based on the input and output signals and the proper SI algorithms, an identified linear or nonlinear model can be generated to describe the behaviour of the identified object.

Figure 3.2 shows a block diagram of the SI process for generating the linear model from the nonlinear 3 DOF dynamic model. The whole simulation process is completed using Matlab. In the block diagram, the nonlinear model for the ASC system directly implements the nonlinear dynamic model in Equation 3.8. The variable u stands for the input signal that is applied to both the ASC nonlinear model and the desired linear mathematic model. By minimizing the difference between the two output signals (y_1 and y_2) using the proper SI algorithm (i.e. ; least squares method), the coefficients in the mathematic model will be adjusted to best represent the ASC system process.

The coefficients of the nonlinear model (m_{ii} and d_{ii}(i=1,2,3)) implements a set of parameters of a monohull ship which has the length of 32 metres and mass of 118000 kg from book [32] (page 104):

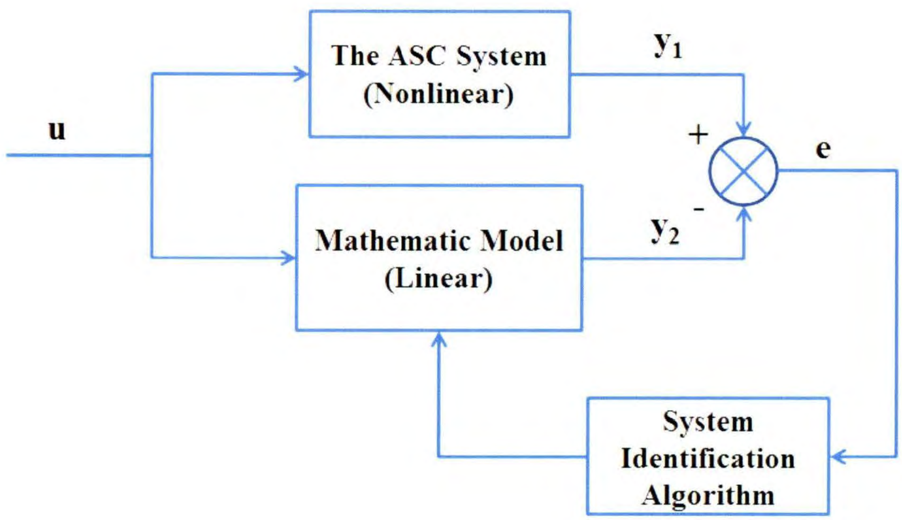

Figure 3.2: SI for the ASC

$$\begin{cases} m_{11} = 120 * 10^3 kg \\ m_{22} = 177.9 * 10^3 kg \\ m_{33} = 636 * 10^5 kgm^2 \\ d_{11} = 215 * 10^2 kg/s \\ d_{22} = 117 * 10^3 kg/s \\ d_{33} = 802 * 10^4 kgm^2/s \end{cases} \tag{3.15}$$

The model used in the SI actually includes two control inputs: the surge force and yaw moment. To get enough information from this nonlinear model, two proper input signals are chosen to excite the system dynamic characteristics. Here the pseudo random binary sequence (PRBS) signals are chosen, and these two signals' range has to be determined according to their physical meanings. The following Matlab codes were used. In this code, u1 stands for the surge force in Newton, while u2 stands for the steering torque in Newton metres.

$\%Matlab\ codes\ for\ generating\ the\ two\ excitation\ signals$

$num = 3000;$

$\%PRBS\ input\ signal\ generater$

$u1 = idinput(num,'prbs', [0, 0.02], [-20,000, 20,000]);$

$u2 = idinput(num,'prbs', [0, 0.008], [-2,000,000, 2,000,000]);$

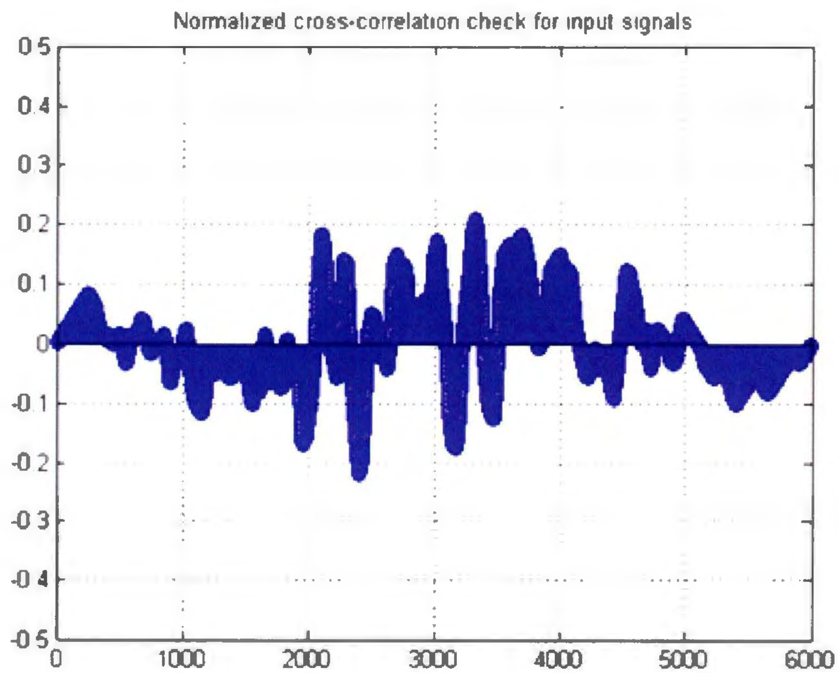

Figure 3.3: Normalized cross-correlation check for the designed two input signals

For each step input, it will take around 50 seconds for the output signal to achieve the steady state, so the minimum interval chosen for the designed PRBS signals is 50 seconds. The cross-correlation analysis of the two input signals are shown in Figure 3.3.

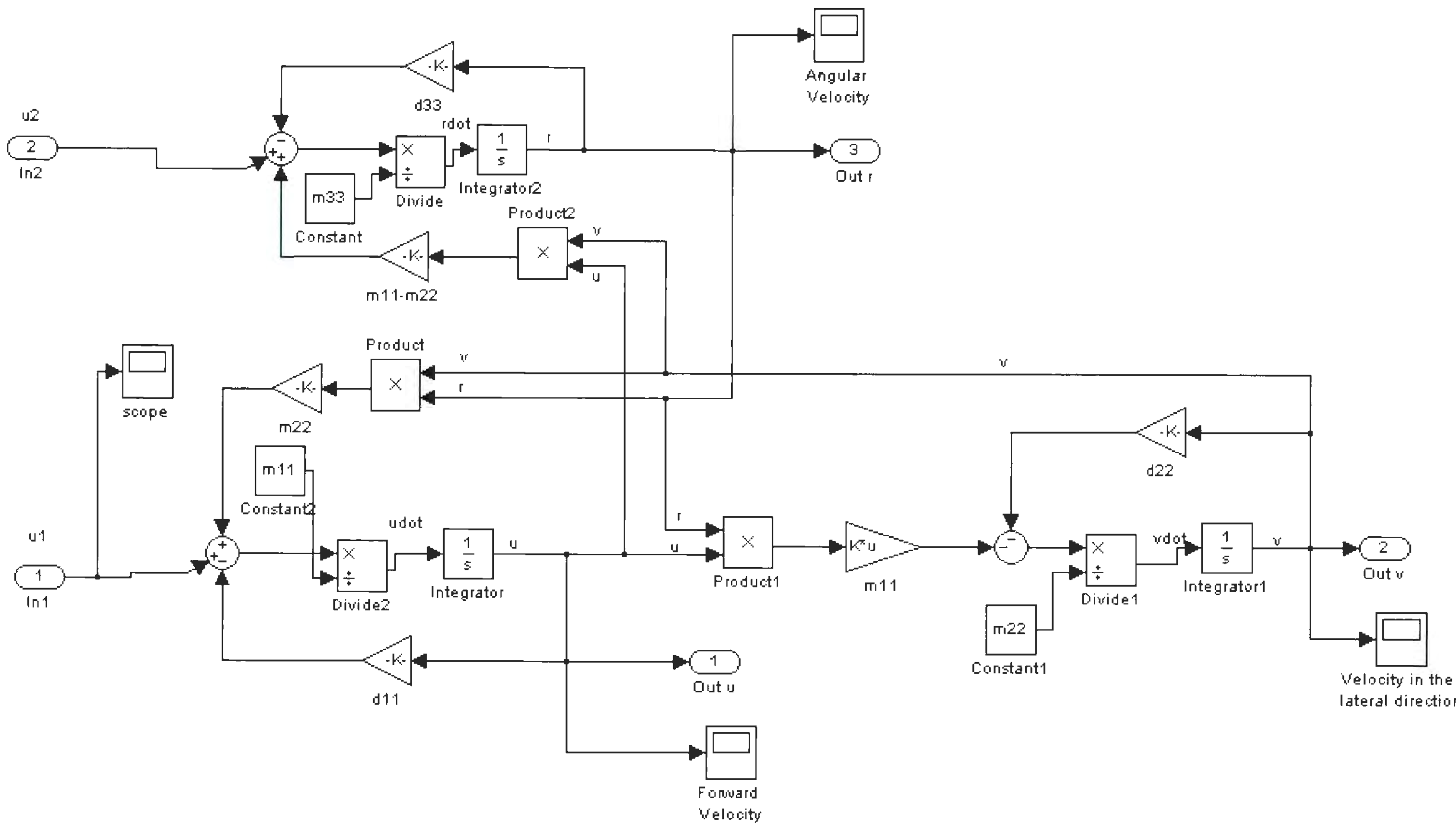

Figure 3.4: Nonlinear model for SI

The correlation between the two input signals are within the acceptable small range of [-0.2, 0.2] in the normalized scale, so the two input signals are uncorrelated and they can be used for the SI simulation.

The nonlinear ASC system model has been built using the Matlab Simulink functions. Final realization of the 3 DOF dynamic model is shown in Figure 3.4. After this, the designed input signals are applied to this model, and the corresponding output data are recorded and shown in Figure 3.5, 3.6 and 3.7.

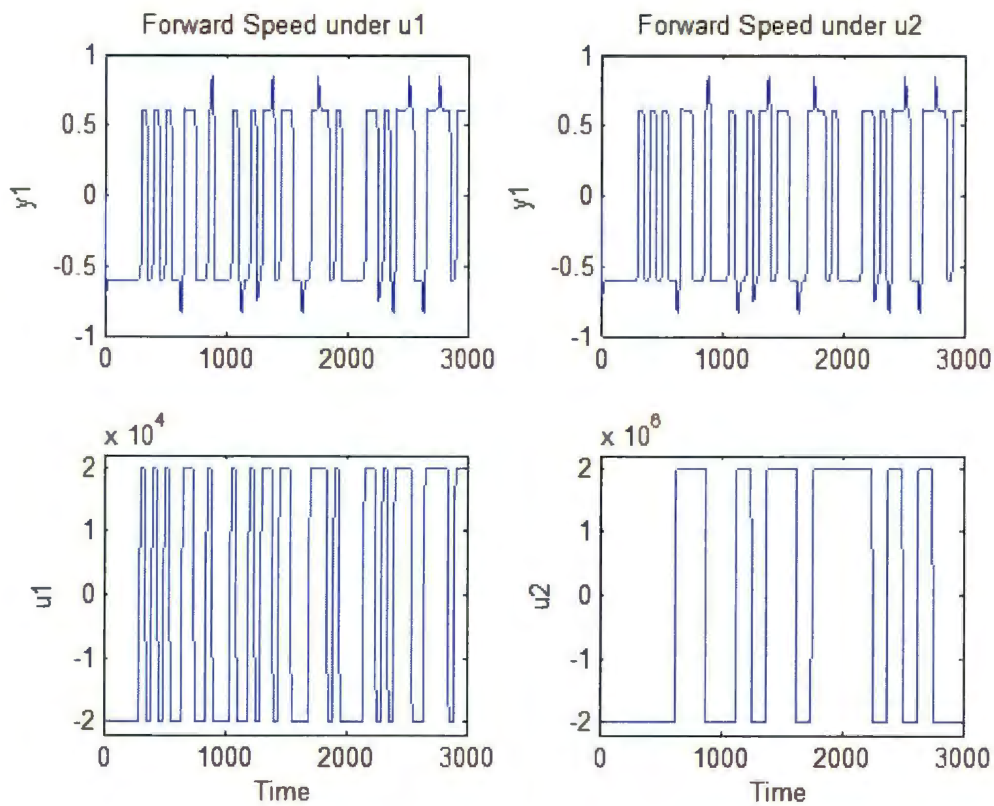

Figure 3.5: Surge velocity under the proposed PRBS input excitation signals

The terms used in Figure 3.5, 3.6 and 3.7 are concluded as follows: u1 stands for the surge force, u2 represents the steering torque, y1 is the surge velocity, y2 is sway velocity and y3 is yaw angular velocity. In each figure, two input signals are plotted with the corresponding output signals.

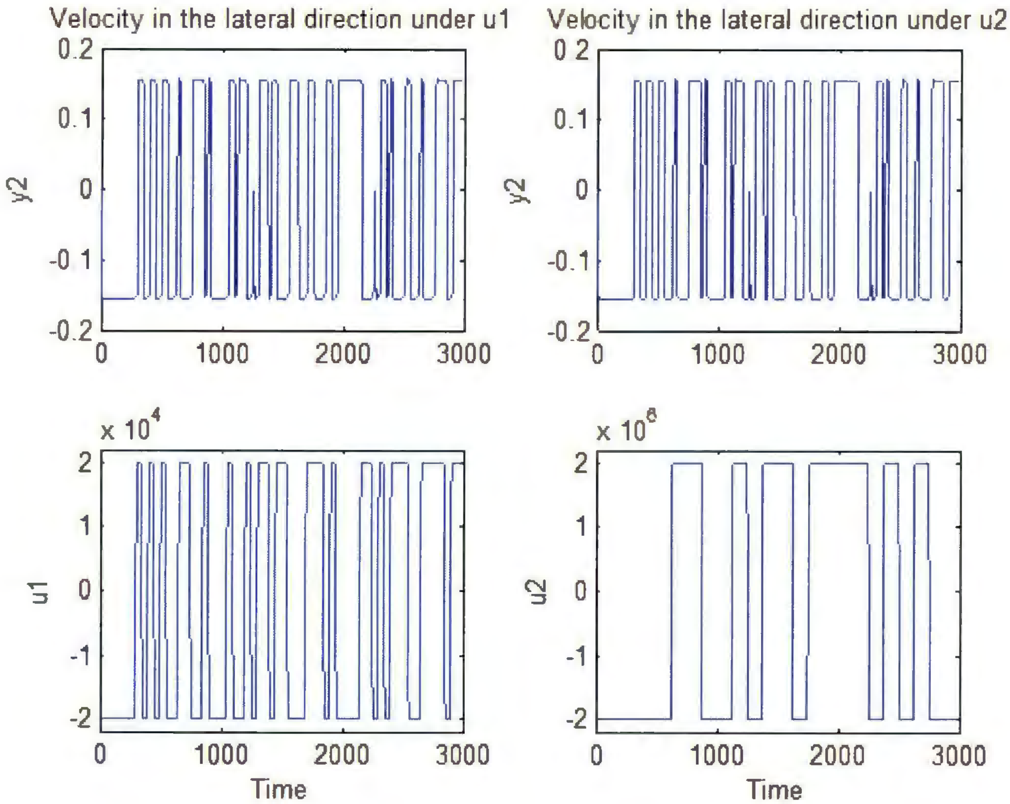

Figure 3.6: Sway velocity under the proposed PRBS input excitation signals

In Figure 3.5, it can be concluded that surge velocity is more correlated with the surge force, because it generally follows the u1 input, while in Figure 3.7 the yaw angular rate follows the steering torque. It seems that the sway velocity follows both u1 and u2 input, which makes sense because the model used does not have direct sway control input. However, Figure 3.6 also indicates that the influence of the two inputs on y2 is smaller than on y1.

After the data are acquired, SI can be performed using Matlab identification toolbox. Since this toolbox can only solve multiple input and single output (MISO) problem, each output signal is used to generate its own identified model. The process of gener-

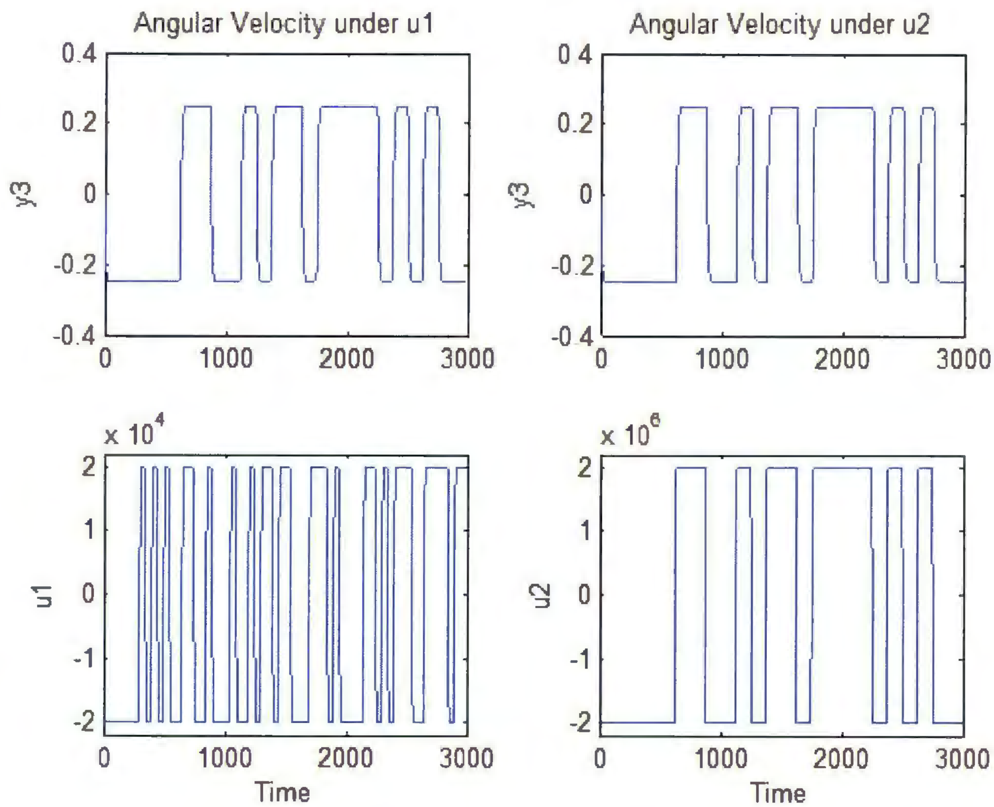

Figure 3.7: Yaw angular rate under the proposed PRBS input excitation signals

ating the three output signal models is similar, so subsequently only the procedures for getting the surge velocity linear model are introduced.

The surge velocity data are imported into the SI toolbox, and then the data are detrended. In this SI process, the test and validation data range is defined as [0, 2000] and [2001, 3000]. The process time delay is determined using the following Matlab codes, and the results are plotted in Figure 3.8.

%Matlab codes for estimation of the process time delay

$u1 = u(:, 1);$

$u2 = u(:, 2);$

$dat1 = iddata(y1, u1); cra(dat1);$

$dat2 = iddata(y1, u2); cra(dat2);$

From Figure 3.8, it can be determined the correlation between output y1 and u1 is more than that of y1 and u2, which complies with the analysis of Figure 3.5. This also implies that it might be possible to remove the u2 input in the generated model. To find a proper model order for the system, the "Linear Parametric Models" function is used. The "Order Selection" feature can help to find a proper model order. After the SI process, an autoregresive model with external input (ARX) was generated as shown in Equation 3.16, this model has been validated using the SI toolbox "Model output" function and it can reach 90.18% best fits (the signal with spikes is the original signal) as shown in Figure 3.9.

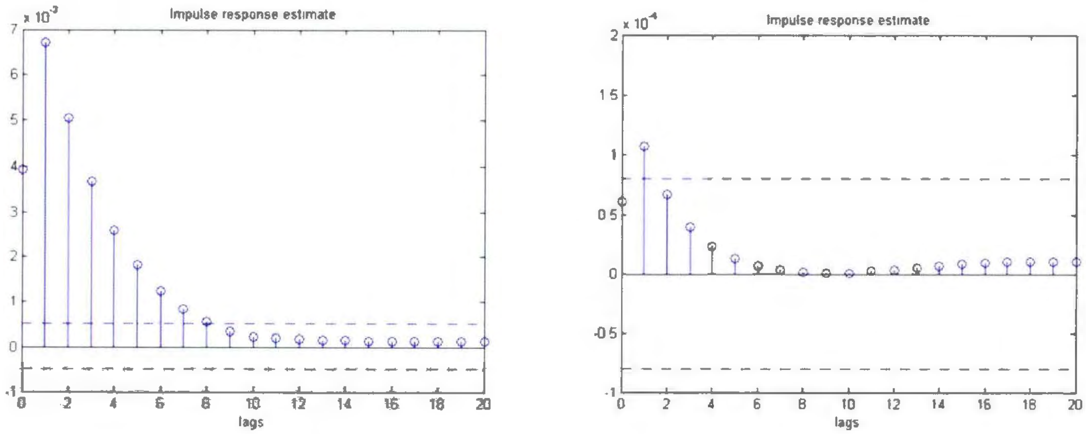

Figure 3.8: Process time delay analysis for input u1 and u2 with output y1

In this ARX model, y(t) is output surge velocity, u(t) is input surge force and steering torque, and e(t) stands for noise. It is clear that in this model B_2 is much smaller than B_1, which validates that the steering force has much less effect on the surge velocity than the surge force.

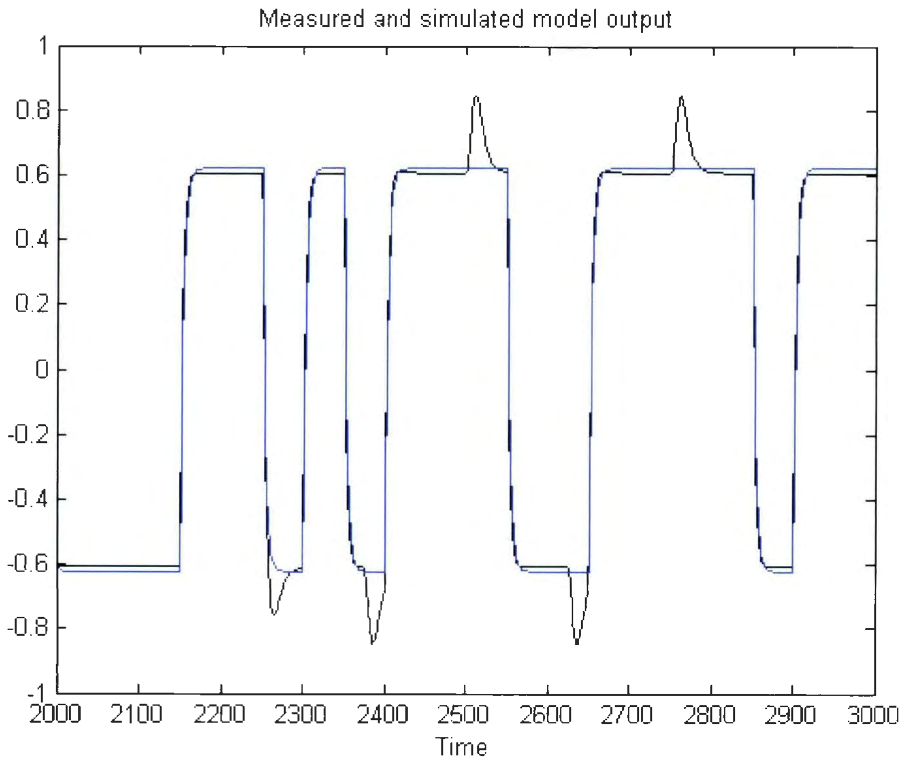

Figure 3.9: Measured and simulated model output comparison

$$
\begin{cases}
A(s)y(t) = B(s)u(t) + C(s)e(t) \\
A(s) = s + 0.2817 \\
B(s) = [B_1(s)\ B_2(s)] \\
B_1(s) = 8.757e - 006 \\
B_2(s) = -1.649e - 010
\end{cases}
\tag{3.16}
$$

Until the simulation stage, since the hydrodynamic coefficients of the ASC were not available, the monohull ship hydrodynamic coefficients were chosen to perform the initial System Identification tests. Through the simulation, it was validated that a proper order linear model that described the straight-line moving behaviour of a monohull ship was generated. The SI used here is a black box identification method, so there is no need to identify all the hydrodynamic coefficients to obtain the linear model as indicated in Section 3.2.1.

Though the simulation was performed on a large ship model, the SI procedure is the same when we perform the SI tests on the ASC. In Chapter 4, a linear second order model for the ASC will be generated using the introduced SI procedures in Section 3.2.2.

Chapter 4

Evaluation of the Autonomous Surface Craft

4.1 The ASC Initial Test

An initial test has been set up as shown in Figure 4.1 to validate the proper functionality of the designed ASC under the supervisory commands from the dock-side computer. The tested functions include proper control from both dock-side computer GUI and the hand controller, proper display of sensor data and the quasi real-time display of the global position of the vehicle in Google Earth software.

Although the control, data display and data logging functions were working fine, when performing the endurance test, it was found that the system would crash after 5 minutes. When a system crash happened, the GUI indicated the required messages could not pass the checksum check. It took some time to figure out this problem, but finally the problem was addressed by adding the wireless modem serial port "flush" function for both sides (the ASC-side and the dock-side computer). The GUI running on the dock-side computer was modified to include the timeout function, so when data

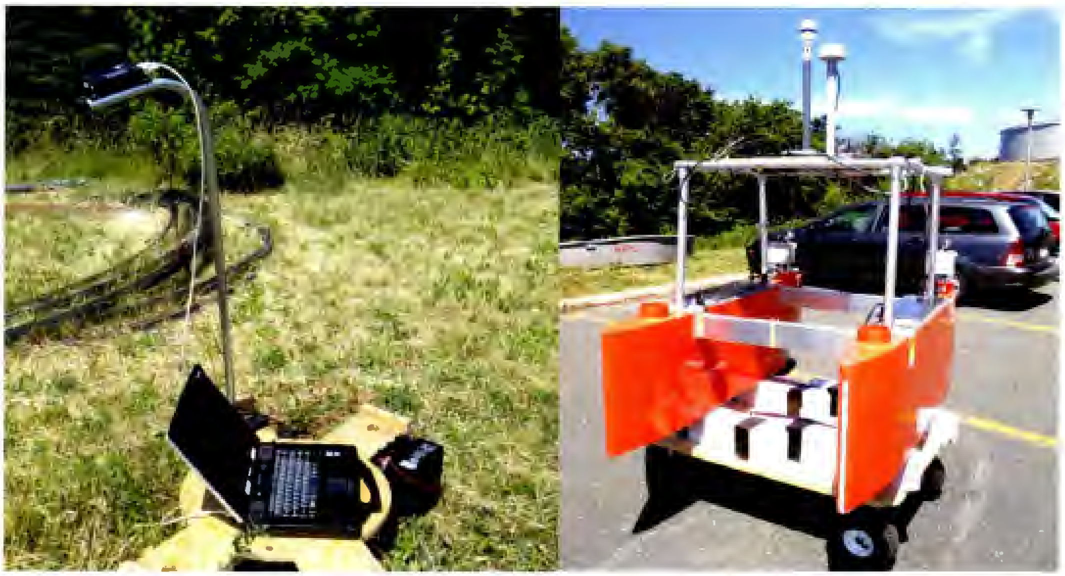

Figure 4.1: The ASC initial test performed outside the Engineering Building

were not received in the desired time, the GUI would move to the next mission. After this modification, the whole system functioned well with no wireless communication errors after a 30 minutes endurance test.

4.2 The ASC Tow Tank Tests and Validation

Tow tank tests were carried out to measure the ASC hull resistance and to qualify the propulsion system. Taking advantage of the size and weight of the designed ASC, it was possible to perform full-scale tests and the results are shown in the following sections. Figure 4.2 shows the experimental setup for the resistance and self-propulsion test. In these tests, the vehicle superstructure is removed, and the tow post is fixed to an adapter plate right in the center of the ASC. The tow post is mainly used for measurement of the towing force along the x axis; however, since it can also move along the z axis and rotate around the y axis, the heave and pitch motion of the vehicle can also be recorded during the tow tank test. To validate the

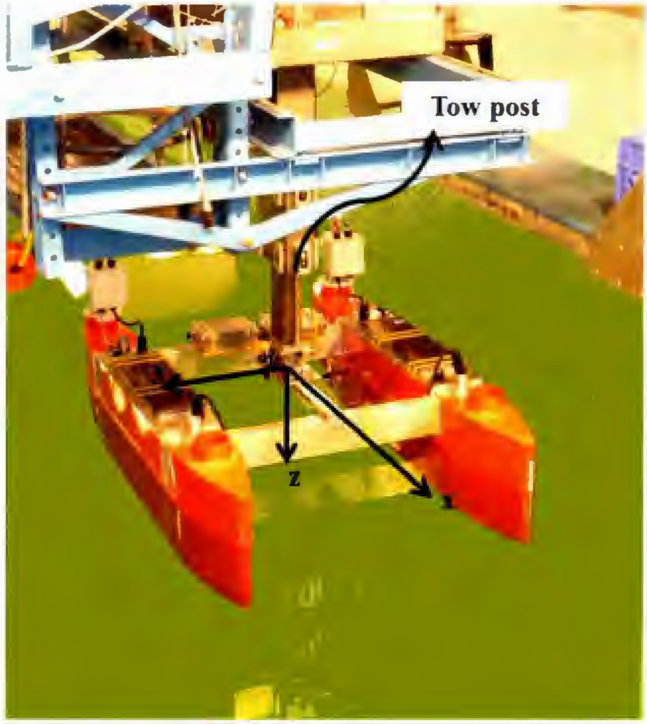

Figure 4.2: Experimental setup for tow tank test

tow tank test results, sea trials have been carried out in Holyrood Arm, Conception Bay, NL.

4.2.1 Resistance Test and Results

For the resistance test, the propellers were replaced by two blade-less nosecones to remove the propellers' induced drag. When performing this test, the vehicle had an initial 0 m/s speed, and then it was towed to the predefined moving speed. This moving speed was maintained for a while for data recording before being reduced to a full stop.

The ASC Froude number can be calculated using Equation 4.1.

$$F_r = \frac{U}{\sqrt{L * g}} \tag{4.1}$$

where U is the vehicle advance moving speed in m/s, L represents the length of the submerged portion of the vehicle and g stands for the gravitational constant.

The surface vessel performance with respect to its Froude number is given by Equation 4.2.

$$F_r = \begin{cases} < 0.4 - 0.5 \ (\textit{displacement mode}) \\ \quad 0.5 - 1.0 \ (\textit{displacement and planing mode}) \\ > 1.0 - 1.2 \ (\textit{planing mode}) \end{cases} \tag{4.2}$$

To maintain the ASC in displacement mode, the Froude number has to be less than 0.5 (dimensionless). The length of the ASC submerged portion is measured as 1.5 m, and it is assumed that g is 9.81 m/s^2. The speed range of the vehicle is calculated to be less than 1.53 m/s. Therefore, during the resistance test, the maintained speed range is defined to be from 0.3 m/s to 1.3 m/s at a step of 0.1 m/s, so a total of 11 experiments were required to be performed. In each experiment, the towing force, heave movement and pitch angle were recorded. Since the sampling period for each variable is 0.00062 s, a moving average filter was implemented to remove the noise issue from the measured data. A conclusion of the filtered towing force and calculated drag with respect to the ASC moving speed is provided in Table 4.1.

Figure 4.3 shows the drag speed curve from the resistance test, and as shown, the x axis represents the ASC advance speed, while the y axis is the measured towing force that is equal to the vehicle drag. Error bars are added to each measured point to indicate the measurement deviation. It can be seen that the plot is close to a quadratic curve.

The drag of the ASC is mainly contributed by the form drag. The form drag formula as shown in Equation 4.3 can be used to calculate the drag coefficient of the ASC. The variables used in this equation are defined in Table 4.2.

Table 4.1: The ASC resistance test results

Speed(m/s)	Tow Force(N)	Tow Force Offset(N)	Drag(N)	Errorbar
0.3	1.3555	0.162	1.1935	0.4033
0.4	2.4348	0.0495	2.3853	0.5219
0.5	3.6728	0.1377	3.5351	0.55
0.6	5.162	0.1696	4.9924	0.8185
0.7	6.77	0.1377	6.6323	0.5135
0.8	9.1244	0.1629	8.9615	0.5721
0.9	12.3455	0.0477	12.2978	0.5341
1.0	15.8575	0.0782	15.7793	0.4784
1.1	23.135	0.1676	22.9674	0.7932
1.2	36.6207	0.1662	36.4545	1.356
1.3	46.8641	0.1508	46.7133	1.1388

$$Drag = 2 \cdot \frac{1}{2}\rho A v^2 C_D \tag{4.3}$$

Table 4.2: Variable definition for Equation 4.3

Variable	Description	Unit
Drag	The ASC resistance force	N
A	The reference area for one hull (wetted surface area)	m^2
v	The ASC relative speed to water	m/s
ρ	Water density	kg/m^3
C_D	Dimensionless drag coefficient	-

To calculate the drag coefficient, the values for the remaining variables have to be determined. Drag and moving speed v are already provided in Table 4.1, and the water density ρ is 1000 kg/m^3. However, it is found that when the vehicle speed is increased, there is an additional pitch angle and heave movement, which will affect the reference area A. Therefore, the reference area has to be calibrated using the pitch and heave measurement before calculating the C_D value. Equation 4.4 is used for calculating the new reference area A^*, and the unit used in this equation is metres. In this equation, 0.75 m is the half height of one hull, and 0.37 m is the measured draft of the vehicle under the tow tank test conditions. In addition to that, the width of each hull is measured as 1.7 m.

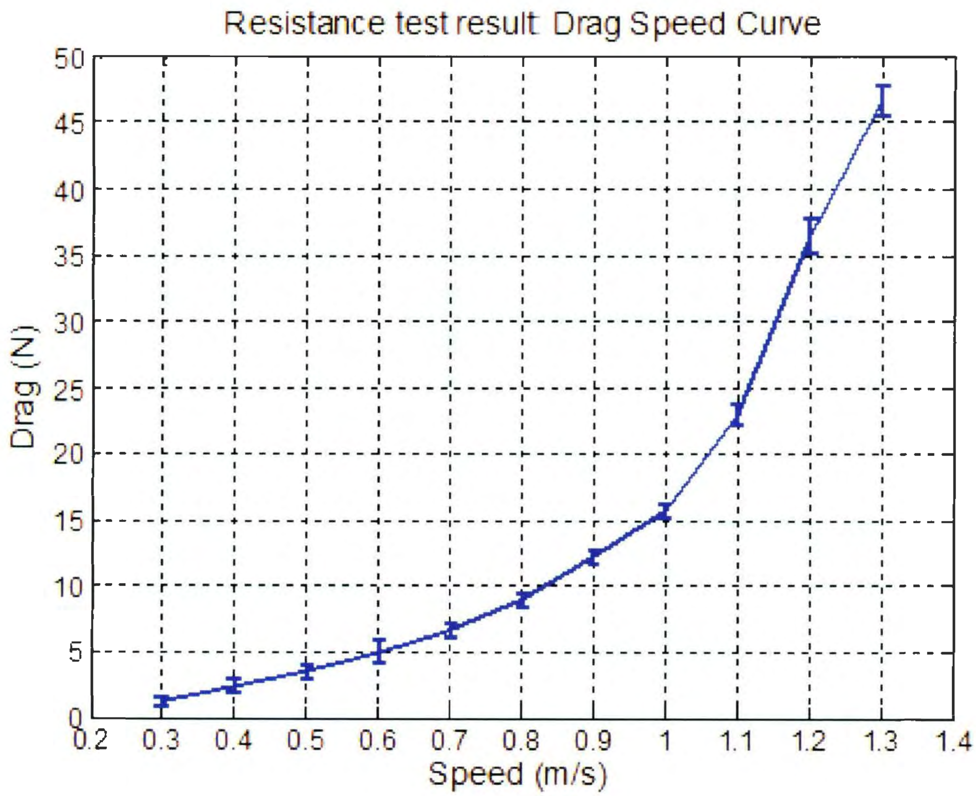

Figure 4.3: Resistance test: drag speed curve

$$A^* = (sin(Pitch) * 0.75 + Heave + 0.37) * Width \tag{4.4}$$

The calibrated reference area under each moving speed has been calculated and shown in Table 4.3, and C_D is generated. Figure 4.4 shows the plot of the drag coefficient, where the x axis is vehicle advance moving speed and the y axis is C_D.

Table 4.3: Pitch angle in the resistance test

Speed(m/s)	Pitch(degree)	Heave(m)	Reference Area(m^2)	C_D
0.3	0.9193	1.3875×10^{-3}	0.064448382	0.205763
0.4	1.3661	2.0238×10^{-3}	0.065549707	0.227432
0.5	1.1189	2.4213×10^{-3}	0.065067716	0.217318
0.6	1.5789	3.7385×10^{-3}	0.066313962	0.209123
0.7	1.3748	3.8858×10^{-3}	0.065885324	0.205437
0.8	1.9036	4.7542×10^{-3}	0.067208112	0.208343
0.9	1.7994	6.6601×10^{-3}	0.067300285	0.225593
1.0	2.3851	8.0009×10^{-3}	0.068829441	0.229252
1.1	2.5521	9.786×10^{-3}	0.069503632	0.273098
1.2	3.6456	13.0193×10^{-3}	0.072480604	0.349274
1.3	4.2284	16.102×10^{-3}	0.074297001	0.372034

As shown in Figure 4.4, C_D stays almost constant within the speed range of 0.3 m/s to 1.0 m/s, but features a rapid increases after 1.0 m/s moving speed. The reason for this big change is because when the advance moving speed of the vehicle is over 1.0 m/s, the pitch angle and heave movement of the vehicle becomes larger, and the water starts to overflow the bow of the vehicle.

From this resistance test, the ASC drag coefficient is generated and it stays around 0.23 within the speed range of 0.4 m/s to 1.0 m/s. In the following self-propulsion test, the same vehicle moving speed range is chosen.

4.2.2 Self-propulsion Test and Results

For the self-propulsion test, the propellers were installed on the vehicle. The two propellers were configured to maintain the same constant rotational speed, but different

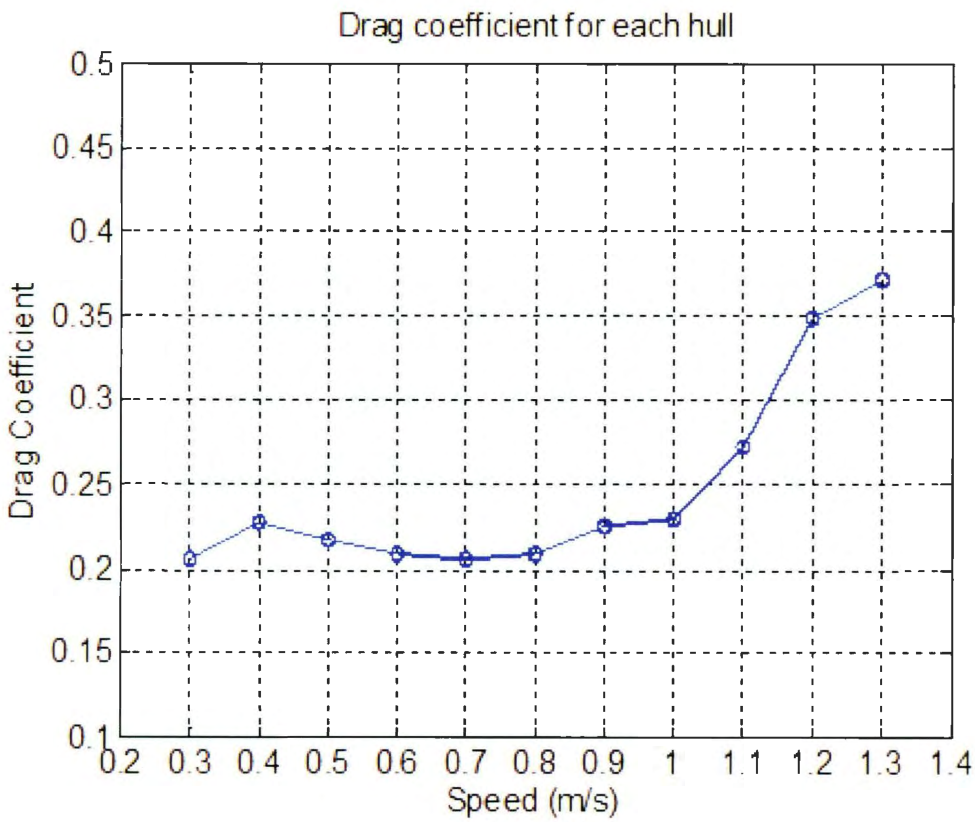

Figure 4.4: Drag coefficient

rotational speeds were used for each test when the vehicle was towed to the predefined moving speed with the towing force recorded.

Since the speed range chosen for this test is from 0.4 m/s to 1.0 m/s at the step of 0.1 m/s, seven groups of experiments, among which each group corresponds to a maintained moving speed, have been performed. To find the self-propulsion point for each speed condition, inside one experimental group, the propeller rotational speed is varied from low to high to change the vehicle status from under-propelled to over-propelled.

The results of this test are shown in Figure 4.5. In Figure 4.5, x axis stands for the two propellers rotational speed, while y axis is the recorded tow force. Each speed condition features a specific line marker, and it can be seen that 5 sets of tests are performed for each moving speed conditions. The self-propulsion point under each speed is defined as the intersection of each curve with the line that indicates that the measured tow force is zero. The self-propulsion point implies that if the vehicle is propelled by the specified rotational speed, it will reach the corresponding final steady state moving speed.

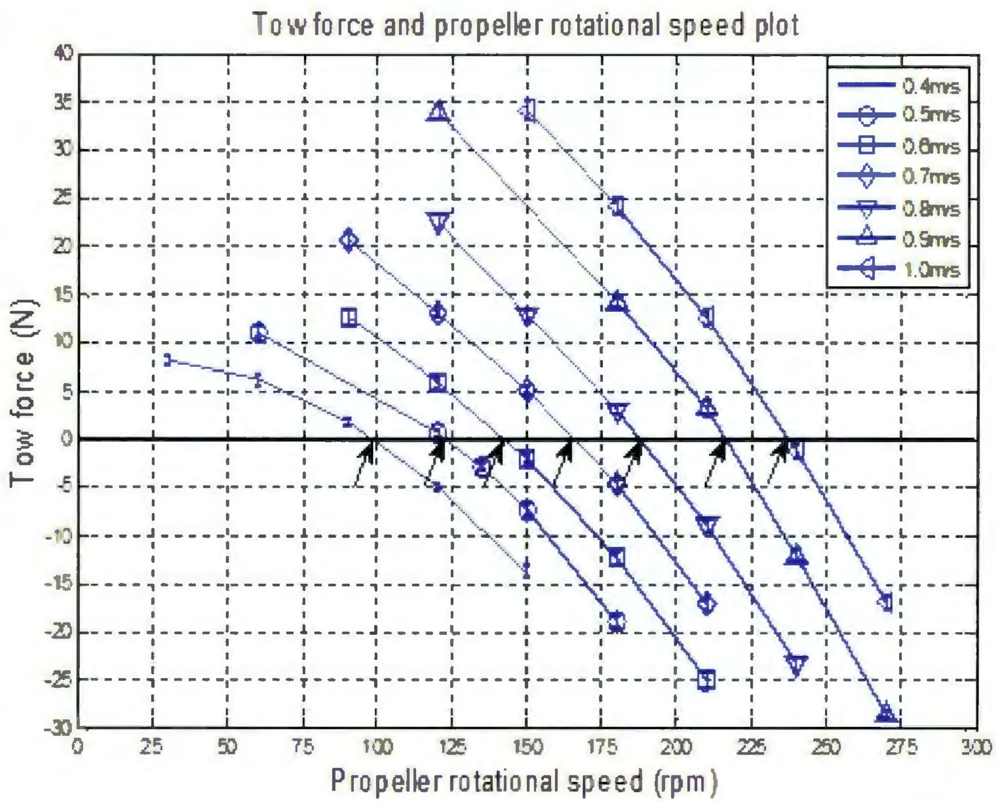

Figure 4.5: Self-propulsion test results under different moving speed conditions

Table 4.4 shows a summary of the self-propulsion points. These points are plotted out as shown in Figure 4.6, and it indicates a linear relationship between the vehicle moving speed and the propeller rotational speed. Therefore, a least square curve fitting is performed, and the fitted curve plotted in dotted line has quite small residuals compared with the measured data. The generated Equation 4.5 can also be used to estimate the self-propulsion points beyond the self-propulsion test speed range.

Table 4.4: Self-propulsion points conclusion

Speed (m/s)	Propeller (rpm)
0.4	99
0.5	122
0.6	142
0.7	166
0.8	188
0.9	216
1	237

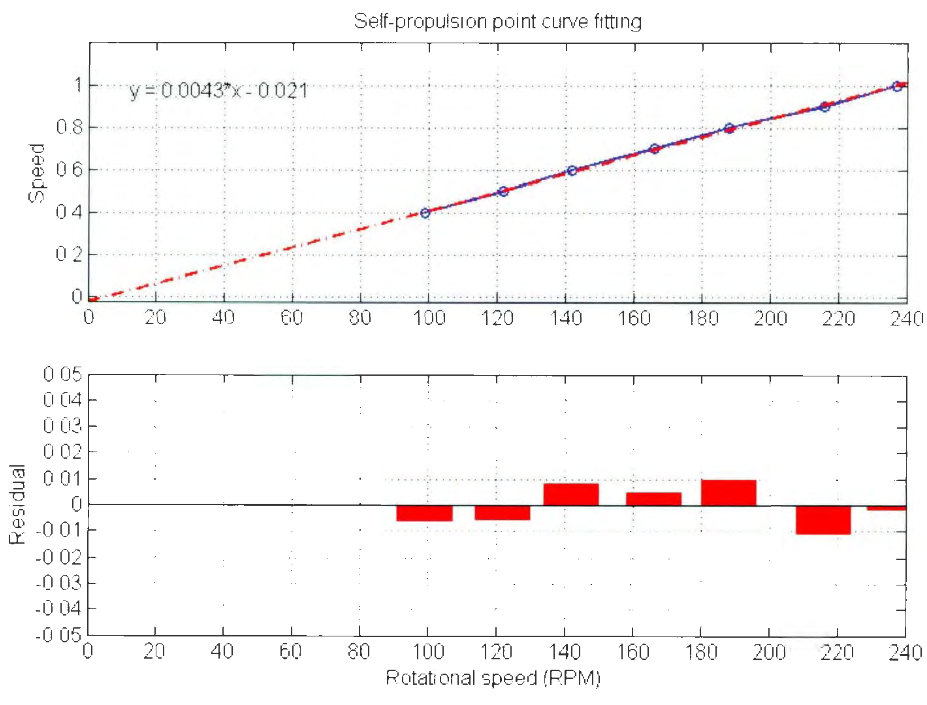

Figure 4.6: Self-propulsion points curve fitting

$$y = 0.0043 * x - 0.021 \tag{4.5}$$

4.2.3 Propulsion Model

By combining the results from the resistance and self-propulsion tests, a propulsion system model can be generated.

In the self-propulsion tests, Equation 4.6 is established. In this equation, towing force is directly measured by the tow post and ASC drag can be achieved from the resistance test, so the propulsion system thrust value can be calculated.

$$Tow\ Force = Thrust - Drag \tag{4.6}$$

Thrust value (two propellers) is obtained and replotted in Figure 4.7. In Figure 4.7, the x axis is the two propellers rotational speed squared, while the y axis is the calculated propulsion system thrust value. Each speed condition features a specific line marker. Although the ASC advance moving speed is changed, each line features almost the same slope.

From Figure 4.7, it can be concluded that when the ASC moves, the thrust from the propulsion system will be affected by two factors: ASC moving speed and two propellers rotational speed squared. If this relationship is defined as in Equation 4.7:

$$T = f(\Omega^2, V_a) \tag{4.7}$$

where T is the thrust, Ω is the propeller rotational speed and V_a is the advance velocity of the vehicle.

It is possible to generate a model that can properly describe the relationship between the thrust and the two factors Ω and V_a. Equation 4.8 shows the model for parameter

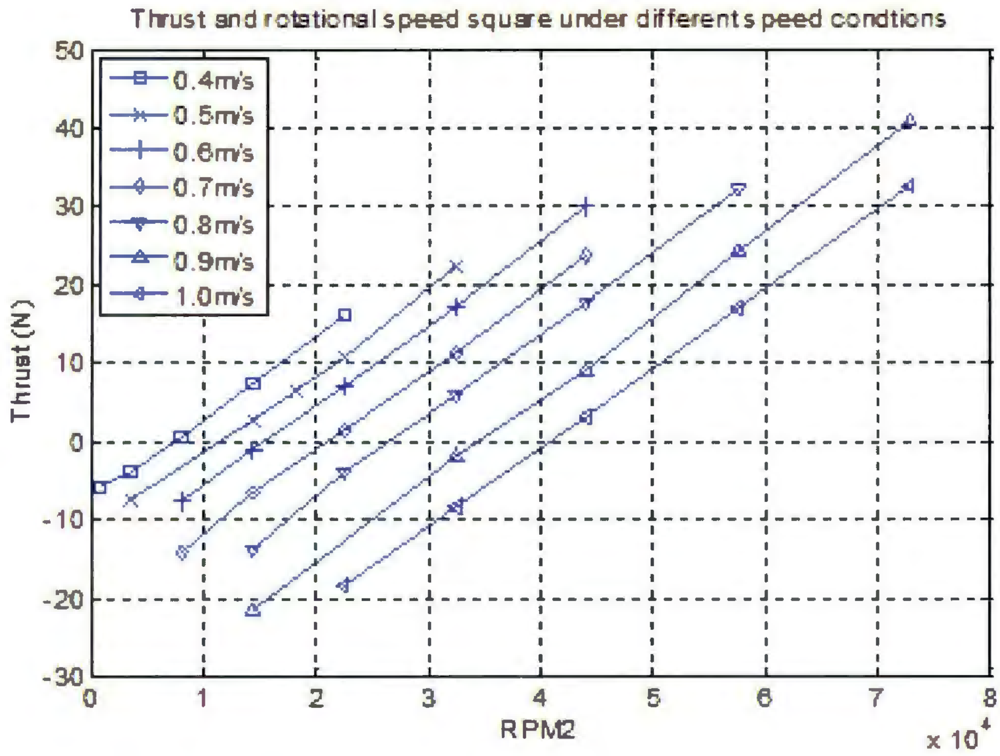

Figure 4.7: Thrust force under different speed conditions

identification, and the used variable definitions are summarized in Table 4.5.

$$T = C_T \Omega^2 + b_0 + b_1 V_a \ (V_a > 0) \tag{4.8}$$

Table 4.5: Variable definition for Equation 4.8

Variable	Description	Unit
T	Thrust force	Unit
Ω	Propeller rotational speed	rpm
C_T	Dimensionless propeller rotational speed squared coefficient	-
V_a	Advance speed	m/s
b_0 and b_1	Dimensionless velocity coefficient	-

Least square curve fitting has been used to identify the coefficients C_T, b_0 and b_1 for this thrust model under the speed range from 0.4 m/s to 1.0 m/s. Finally, the identified propulsion system model is shown in Equation 4.9.

$$T = 0.0010\Omega^2 + 17.7853 - 58.7623V_a \ (V_a > 0) \tag{4.9}$$

4.2.4 Sea Trials and Results

Sea trials have been performed in Holyrood Arm, Conception Bay, NL, to validate the tow tank test results. When performing the sea trials, it was found that in the real sea conditions, vehicle operation status would be affected by wind, currents and waves. In particularly, the heading of the vehicle is easy to be changed by these environmental factors.

In order to validate the tow tank test results, the vehicle has to move in a straight line regardless of the environmental interferences, therefore a heading PI controller has to be implemented. Figures 4.8 to 4.10 show the validation test results with the ASC moving speed range from 0.4 m/s to 1.0 m/s. In each figure, the x axis indicates time, while the y axis includes the information of ASC moving speed, two propellers rotational speed and the ASC heading.

In each test, the propellers rotational speed is assigned according to the self-propulsion points (Table 4.4) from the tow tank test, and by changing to different self-propulsion points, the ASC will reach different final steady moving speed. As shown in Figure 4.8 to Figure 4.10, the difference between the two propellers rotational speed is introduced by the PI controller to change the ASC heading. The sea trail results are compared with the self-propulsion points as shown in Table 4.6.

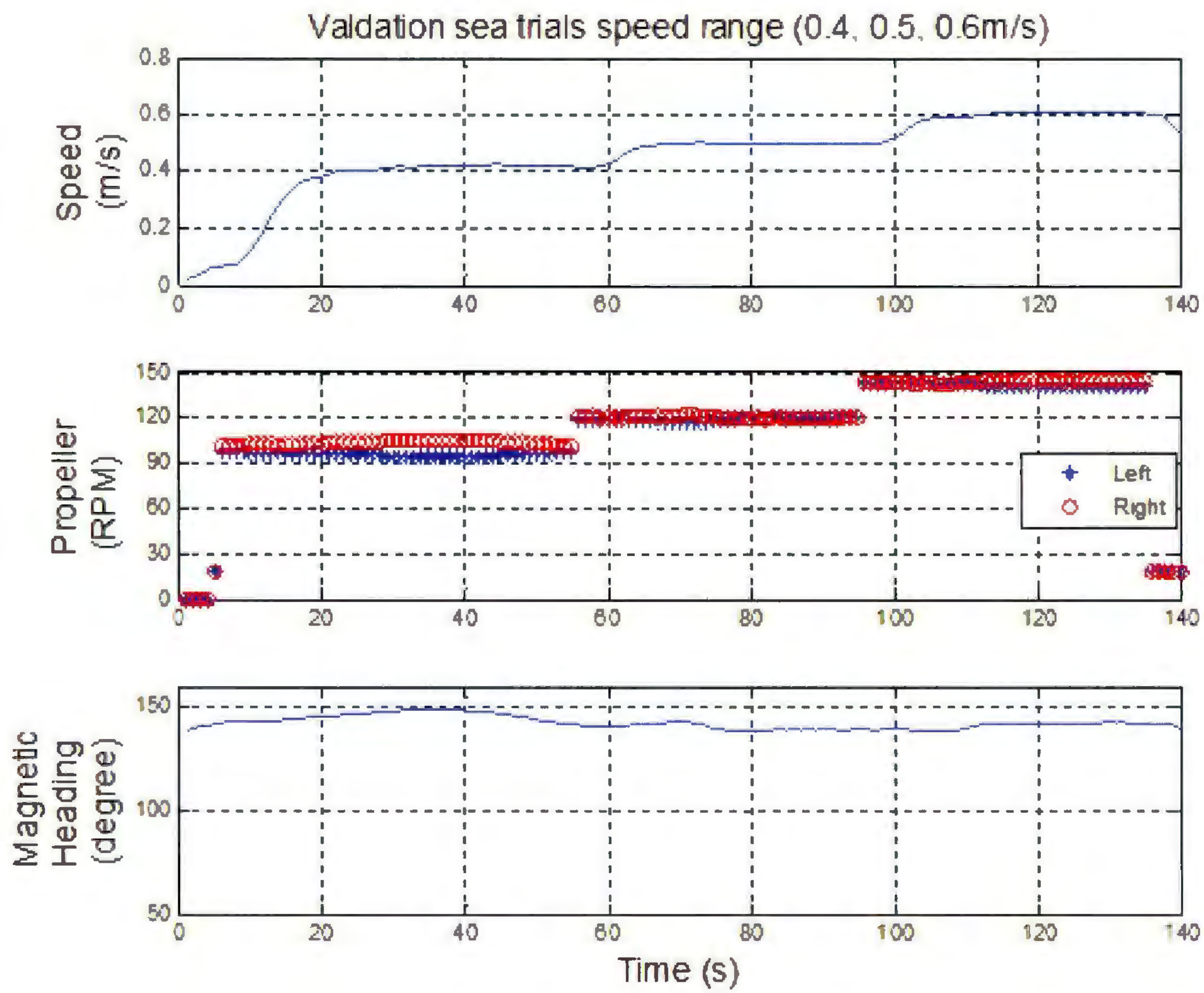

Figure 4.8: Advance speed, propeller rotational speed and magnetic heading with respect to time (0.4 to 0.6 m/s)

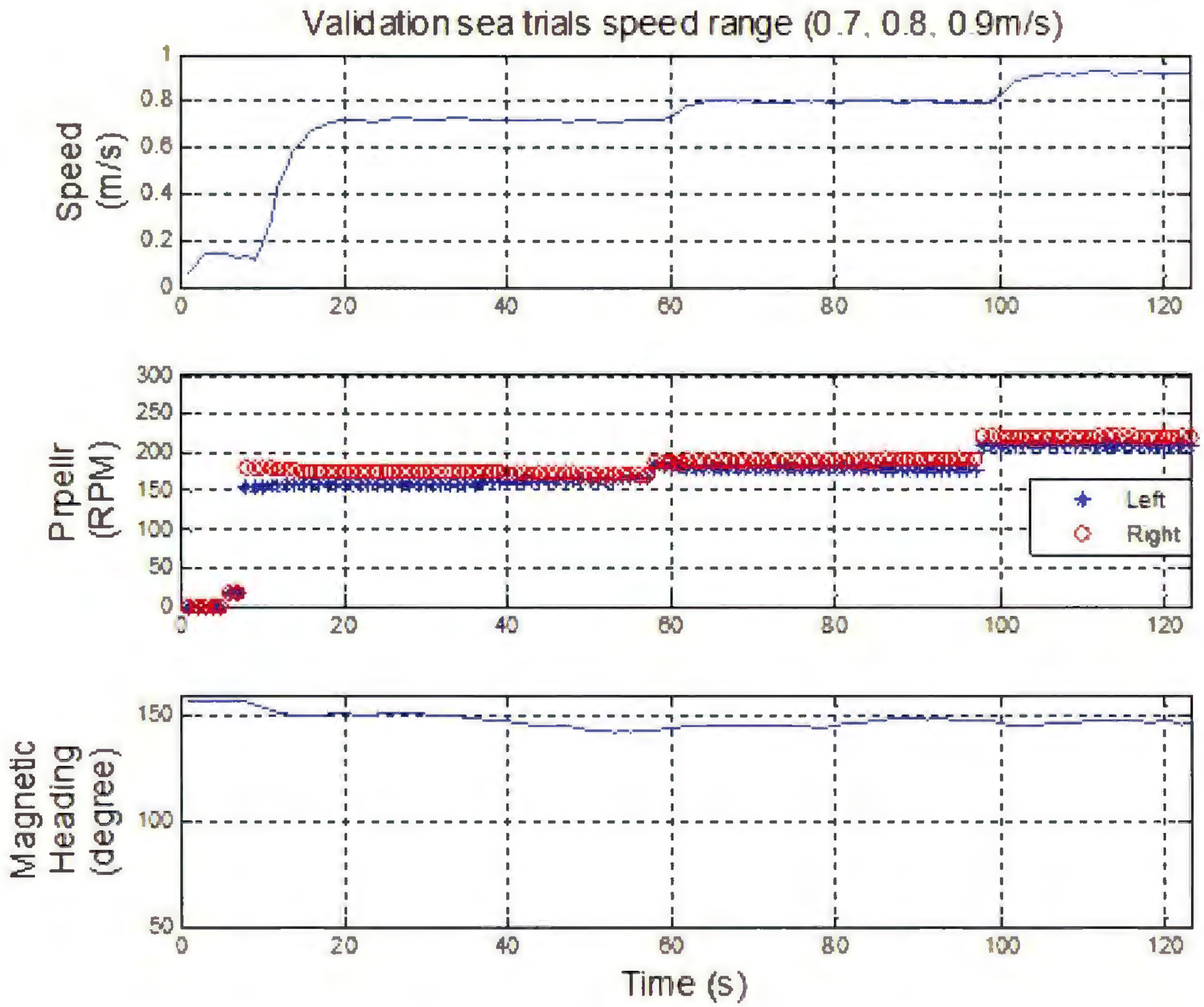

Figure 4.9: Advance speed, propeller rotational speed and magnetic heading with respect to time (0.7 to 0.9 m/s)

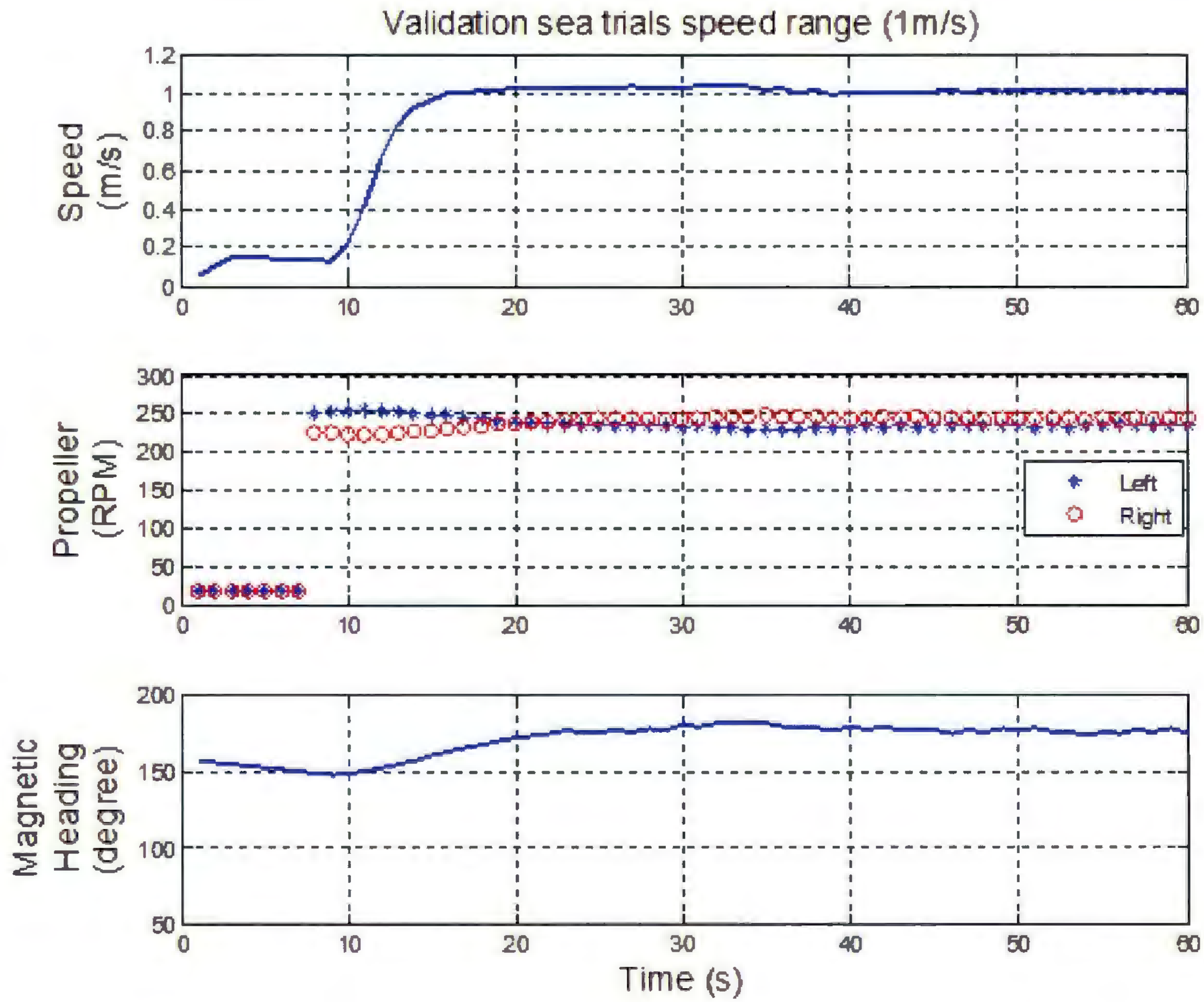

Figure 4.10: Advance speed, propeller rotational speed and magnetic heading with respect to time (1.0 m/s)

It can be concluded that the difference from the two tests is quite small (within 4.7%), and the ASC self-propulsion points are validated. Since the forward speed is measured using the GPS and the uncertainty is 0.1 m/s, the differences seem to be the result of the environmental influences.

Table 4.6: Sea trials results compared with the tow tank test results

-	Tow Tank	Sea Trials	-
Propeller(rpm)	Speed(m/s)	Speed(m/s)	Difference(%)
99	0.40	0.4187	4.68
122	0.50	0.4946	1.08
142	0.60	0.6065	1.08
166	0.70	0.7162	2.31
188	0.80	0.7978	0.28
216	0.90	0.9201	2.23
237	1.00	1.0195	1.95

4.3 The ASC Steering Model

The ASC steering model has been generated using the system identification (SI) technique as discussed in Chapter 3. The steering of the vehicle is realized by applying different rotational speeds to both of the independently controlled propellers. In this modelling process, it is expected to find a relationship between the input, the differential rotational speed, and the heading of the ASC.

If the left and right propeller rotational speed is defined as n_L and n_R, in this experiment, the input of the ASC system is defined to fulfil the conditions as shown in Equation 4.10. By maintaining the summation of n_L and n_R as constant, the vehicle advance moving speed can be regarded as constant. According to Equation 4.6, the steady state moving speed of the vehicle is calculated to be around 0.72 m/s.

$$\begin{cases} n_L + n_R = 336rpm \\ n_L - n_R = \pm 100rpm \end{cases} \tag{4.10}$$

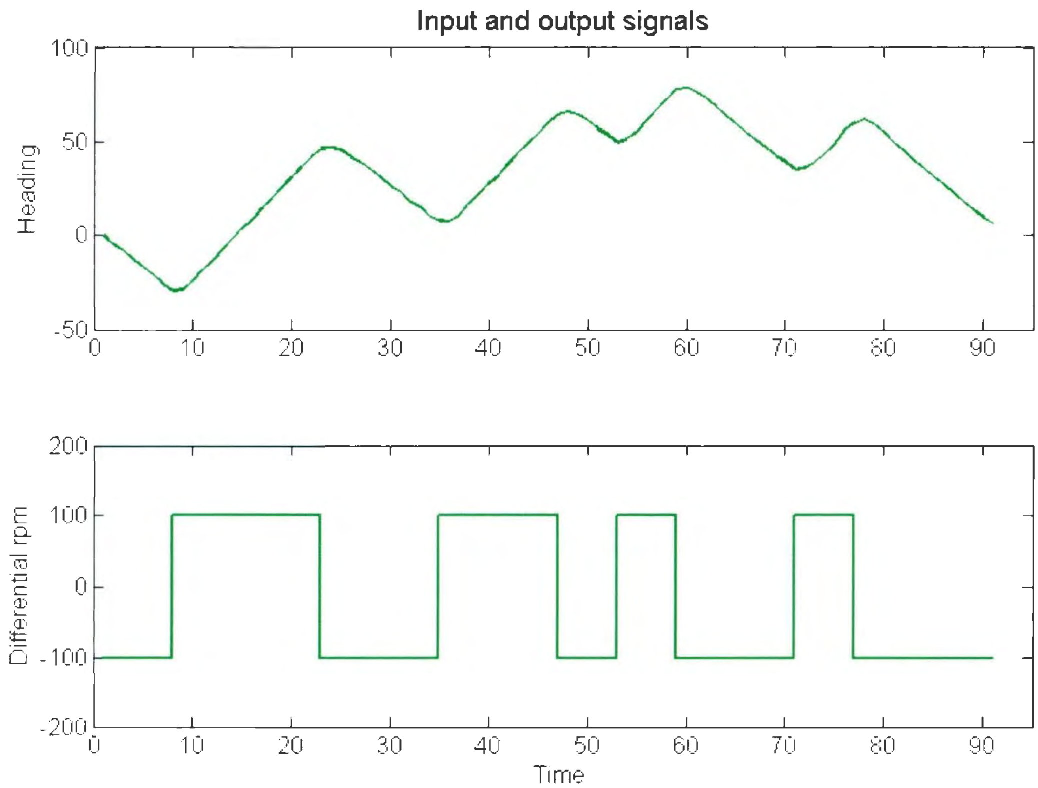

Figure 4.11: Measured ASC system input and output signals

Figure 4.11 shows the imported control input (differential rotational speed $n_L - n_R$) and the measured output heading data from sea trials. In this figure, x axis represents time, while y axis includes the ASC heading and two propellers differential rotational speed. In this time range, the vehicle moving speed is validated to be constant around 0.72m/s.

A linear continuous-time state-space model is expected to be identified based on the recorded data. This desired model is shown in Equation 4.11:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases} \tag{4.11}$$

where u(t) is control input of the differential rotational speed, x(t) is state variables which is a vector including the heading and vehicle turn rate and y(t) is the heading output. A, B, C and D stand for the parameters that are required to be identified. By implementing the SI to this group of data, the value for the parameters A, B, C and D are achieved and shown in Equation 4.12.

$$\begin{cases} A = \begin{pmatrix} 0.01882 & 0.03015 \\ -0.04801 & -0.3997 \end{pmatrix} B = \begin{pmatrix} -0.0001254 \\ -0.000658 \end{pmatrix} \\ C = \begin{pmatrix} -291.5 & 2.414 \end{pmatrix} D = \begin{pmatrix} 0 \end{pmatrix} \end{cases} \tag{4.12}$$

Therefore, a transfer function form ASC steering model can be generated as shown in Equation 4.13.

$$\begin{cases} A(s)y(t) = B(s)u(t) + C(s)e(t) \\ A(s) = s^2 + 0.3809s - 0.006075 \\ B(s) = 0.03497s + 0.02044 \end{cases} \tag{4.13}$$

The identified model was validated using the sea trail measured data. As shown in Figure 4.12, a set of sea trial data was extracted from Figure 4.11 (time range from

21 second to 51 second). Then the corresponding input signal was applied to the identified ASC steering model, and the output ASC heading angle was recorded and plotted in Figure 4.12. The best fit (coefficient of determination) was calculated as 88.29%.

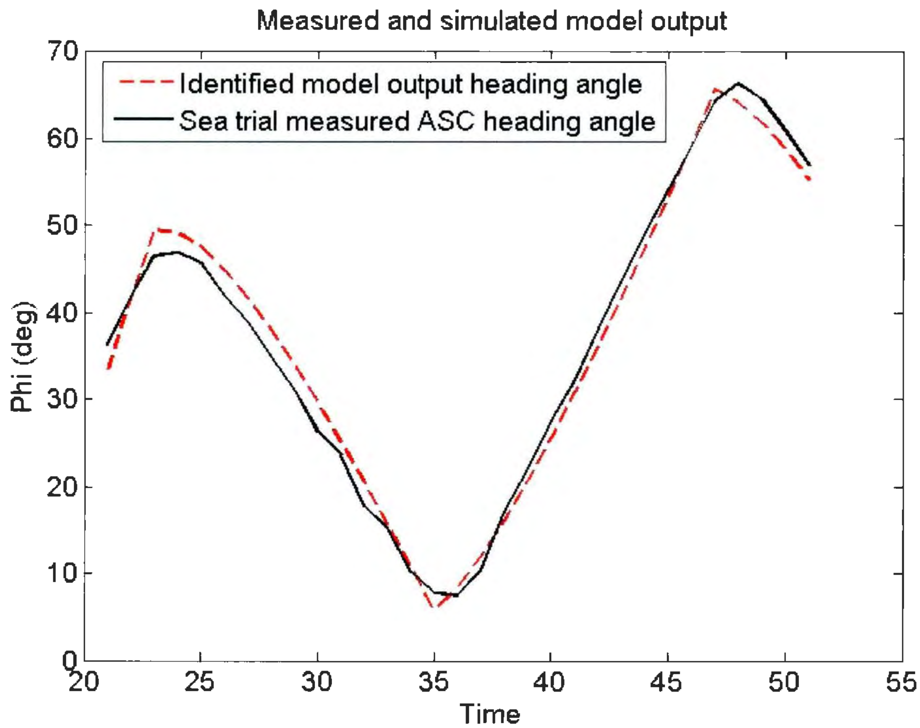

Figure 4.12: Measured data and simulated model output

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The CAN based communication system has been realized and implemented on the developed ASC. Four separate CAN nodes were developed, and they were successfully synchronized using the TRM synchronization mechanism, which was evaluated using the DPO4034 oscilloscope CAN bus trigger function.

The 900 MHz wireless communication link was successfully used in the ASC tests, and the hand controller feature was especially useful when launching and retrieving the vehicle during the sea trials. The developed Matlab GUI was used for data display and data logging, and it was useful for adding more functions, i.e. PID control algorithm, to the system without changing the main program.

A general nonlinear 3 DOF model has been generated in order to describe the motion of a marine vessel in the horizontal plane. Two methods were discussed to be used to get the corresponding linear model. The Taylor series expansion is a common way to linearize a non-linear model around an equilibrium point. However, the generated model only worked near the equilibrium point. In addition, the model coefficients need to be identified using more experiments. The System Identification (SI) technique could be used to get a linear model for a complicated system process. However, it was necessary to find a proper procedure to identify a marine vessel's model, and to decide the required input and output signals and identification algorithms. Therefore. Matlab-Simulink was used to perform this initial SI tests. The monohull ship hydrodynamic coefficients were used in the 3 DOF nonlinear model as the testing model. It was assumed that this nonlinear model can properly describe the monohull ship's motion in horizontal plane. By applying the Pseudo Random Binary Sequence (PRBS) input signals, a linear ship model has eventually been generated. The same identification process has also been used on identifying the steering model of the ASC as stated in Chapter 4.

The ASC hull drag coefficient was generated from the resistance tests, and the vehicle self-propulsion points were obtained and validated by the sea trials results. Based on the tow tank tests data, an ASC propulsion system model was developed. Then the SI was implemented to get the steering model of the ASC, and finally a state space steering model was achieved.

The main contribution of this thesis project was that a CAN-bus based distributed communication and control system was successfully built and used on the developed ASC. In addition, a new weather sensor was successfully integrated into the ASC to provide wind, temperature and barometric data. Moreover, the full-scale ASC resistance test and self-propulsion tests were performed, and the ASC hull drag coefficients and self-propulsion points were acquired. Finally, the proposed SI procedures from simulation part in Chapter 3 were successfully used to obtain a linear steering model of the ASC based on the sea trials data. This linear model will be used in the linear controller design in the future.

5.2 Future Works

A new CAN node is planned to be integrated into the developed CAN-bus based communication system to enable more on-board autonomy of the ASC. More sensors are possible to be connected into the CAN network, so the ASC can perform more sophisticated ocean survey or environmental monitoring tasks.

The ASC launch and recovery are inconvenient during the sea trials, so a plan to design a specific ASC trailer cart especially for launch and retrieval of the ASC will be carried out. This trailer cart is still under development, and minor modifications are needed to complete the design.

The generated steering model has to be validated by the open water tests, and a more complete system model that takes into account the environmental interferences will be generated and evaluated.

A high level navigation and control algorithm will be developed and experimented using the designed ASC.

Bibliography

[1] Z. Li, R. Bachmayer, "The Development of a Robust Autonomous Surface Craft fro Deployment in Harsh Ocean Environments", NECEC conference, 2012.

[2] J. Manley, "Unmanned Surface Vehicles, 15 Years of Development", Proceedings Of Ocean'08 MTS/IEEE, 2008.

[3] J. Manley, "Development of the Autonomous Surface Craft ACES", Oceans '97 MTS/IEEE Conference Proceedings, October, 1997.

[4] C. Goudey, T. Consi, J. Manley, M. Graham, B. Donovan, L. Kiley, "A Robotic Boat for Autonomous Fish Tracking", Marine Technology Society Journal Vol. 32 No. 1, Spring 1998.

[5] J. Manley, A. Marsh, W. Cornforth, and C. Wiseman, "Evolution of the Autonomous Surface Craft AutoCat", Proceedings of Oceans 2000, MTS/IEEE Providence, RI, October, 2000.

[6] G.N. Roberts, R. Sutton, "Advanced in Unmanned Marine Vehicles", Chapter 16 2006.

[7] A. Pascoal, P. Oliveira, C. Silvestre, L. Sebastiao, M. Rufino, V. Barroso, J. Gomes, G. Ayela, P. Coince, M. Cardew, A. Ryan, H. Braithwaite, N. Cardew, J. Trepte, N. Seube, J. Champeau, P. Dhaussy, V. Sauce, R. Moitie, R. Santos,

F. Cardigos, M. Brussieux, P. Dando, "Robotic ocean vehicles for marine science applications: the European ASIMOV project", OCEANS 2000 MTS/IEEE Conference and Exhibition , vol.1, no., pp.409-415 vol.1, 2000.

[8] M. Caccia, G. Bruzzone, R. Bono, "A Practical Approach to Modeling and Identification of Small Autonomous Surface Craft", Oceanic Engineering, IEEE Journal of , vol.33, no.2, pp.133-145, April 2008.

[9] W. Naeem, T. Xu, R. Sutton, A. Tiano, "The design of a navigation, guidance, and control system for an unmanned surface vehicle for environmental monitoring," Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment June 1, 2008.

[10] J. Curcio, J. Leonard, A. Patrikalakis, "SCOUT - A Low Cost Autonomous Surface Craft for Research in Cooperative Autonomy", in IEEE Oceans, 2005.

[11] M. Caccia, R. Bono, Ga. Bruzzone, Gi. Bruzzone, E. Spirandelli, G. Veruggio, A.M. Stortini, "Design and Exploitation of an Autonomous Surface Vessel for the Study of Sea-Air Interactions", Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on , vol., no., pp. 3582-3587, 18-22 April 2005

[12] H. Ferreira, C. Almeida, A. Martins, J. Almeida, N. Dias, A. Dias, E. Silva, "Autonomous bathymetry for risk assessment with ROAZ robotic surface vehicle", OCEANS 2009 - EUROPE , vol., no., pp.1-6, 11-14 May 2009

[13] `http://www.auvsi.org/Home/`

[14] `http://www.asvglobal.com/commercial-unmanned-marine-vehicles/asv-6300c`

[15] http://www.asvglobal.com/commercial-unmanned-marine-vehicles/c-cat

[16] J.R. Higinbotham, P.G. Kitchener, J.R. Moisan, "Development of a New Long Duration Solar Powered Autonomous Surface Vehicle," OCEANS 2006 , vol., no., pp.1-6, 18-21 Sept. 2006

[17] http://liquidr.com/

[18] M. Dunbabin, A. Grinham, and J. Udy, "An autonomous surface vehicle for water quality monitoring," in Proc. Australasian Conference on Robotics and Automation, December 2009.

[19] Y. Girdhar, A. Xu, B.B. Dey, M. Meghjani, F. Shkurti, I. Rekleitis, G. Dudek, "MARE: Marine Autonomous Robotic Explorer," Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, vol., no., pp.5048-5053, 25-30 Sept. 2011

[20] S. Howse, M. Goldring and M. Pitcher, "Underwater glider retrieval using an autonomous surface vehicle", OCEANS 2010, vol., no., pp.1-8, 20-23 Sept. 2010.

[21] http://openp2p.com/pub/a/p2p/2001/12/14/topologies_one.html.

[22] http://openp2p.com/pub/a/p2p/2002/01/08/p2p_topologies_pt2.html.

[23] L. A. Luft, L. Anderson and F. Cassidy, NMEA 2000 a digital interface for the 21st century, Institute of Navigation's National Technical Meeting, San Diego, CA, Jan. 2002.

[24] K. Etschberger. "Controller Area Network: Basics, Protocols, Chips and Applications," IXXAT Press, 88250 Weingarten, Germany, 2001.

[25] A. Pietak, M. Mikulski, "On the adaptation of CAN BUS network for use in the ship electronic systems," POLISH MARITIME RESEARCH, vol., no., pp.62-69, 2009

[26] `http://yachtelectronics.blogspot.ca/`

[27] `http://www.airmartechnology.com/uploads/brochures/pb200.pdf`

[28] `http://mbed.org/`

[29] `http://www.u-blox.com/images/downloads/Product_Docs/NEO-6_DataSheet_(GPS.G6-HW-09005).pdf`

[30] T. I. Fossen, Guidance and Control of Ocean Vehicles. New York: Wiley, 1994.

[31] E. Lefeber, K.Y. Pettersen, H. Nijmeijer, "Tracking control of an underactuated ship," Control Systems Technology, IEEE Transactions on . vol.11, no.1, pp. 52-61, Jan 2003.

[32] K. D. Do, J. Pan, Control of Ships and Underwater Vehicles. London: Springer-Verlag, 2009.

Appendix

A.1 The controller CAN node program

The controller CAN node program is provided in the following part.

```
#include "mbed.h"
#include "Functions.h"
#include "XTend.h"
//Initialize the LEDs
DigitalOut led1(LED1);  //CAN sent/received successfully
DigitalOut led2(LED2);  //XTend error indicator
DigitalOut led3(LED3);  //temp use
DigitalOut led4(LED4);  //UTC timeout(blink once every 1.5s)
CAN canpic(p30, p29);
CAN canairmar(p9,p10);
XTend xtend(p13,p14);//Serial 115200bps XTend(p13,p14) tx, rx
Serial pc(USBTX, USBRX); //USBtx, rx
Timer t;      //Count the running time
Timeout tmout1,tmout2,tmout_UTC;
char CAN_msg[8]={0x01,0x00,0x00,0x32,0x01,0x00,0x00,0x32};
unsigned char CAN_data[8]={0x00};
```

```
unsigned char RTR_id[8]={16,17,18,19.20,21,22,23};
unsigned char RTR_NO[8]={1,2,2,2,3,5,2,5};
unsigned char Nav_CAN_msg[88]={0x00};
unsigned char *p_msg=&Nav_CAN_msg[0];
unsigned char Airmar_msg[80]={0x00};
unsigned char motor_CAN_msg[8]={0x00};
unsigned char motor_fail_msg[8]={0x55,0x00,0x00,0x00,0x56,0x00,0x00,0x00};
unsigned char *p_motormsg=&motor_CAN_msg[0];
unsigned char *p_motorfail=&motor_fail_msg[0];
unsigned char UTC_wait=1;
unsigned char GPS_AHRS_on=1;//GPS and AHRS service flag
unsigned char motor_on=1;    //motor service flag
//timeout functions definition
void atmout1()
{
    GPS_AHRS_on = 0;
}
void atmout2()
{
    motor_on=0;
}
void atmout_UTC()
{
    UTC_wait=0;
    tmout_UTC.detach();
```

```
}
//Main Function
int main()
{
    CANMessage tmsg;
    led1=0;led2=0;led3=0;led4=0;
    canpic.frequency(1000000);  //CAN freq configured as 1MHz
    canairmar.frequency(250000);
    init_AF();
    wait(0.5);  //wait 0.5s for the power up of all devices
    while(1)
    {
        if(xtend.runstart==1)
        {
            tmout_UTC.attach(&atmout_UTC,1.5);
            t.reset(); t.start();

            while(!canpic.read(tmsg) && UTC_wait);
            if(UTC_wait==0) //UTC wait time arrives
            {
                UTC_wait=1;
                led4=1;wait(0.2);led4=0;//indicate timeout
            }
            else //CAN message received
            {
                tmout_UTC.detach();
```

```
                    if(tmsg.data[0]==255 && tmsg.data[1]==255 &&
                      tmsg.data[2]==255 && tmsg.data[3]==255)
                    {
                        led3=!led3;wait(0.1);led3=!led3;
                        systemrun(&tmsg.data[0]);
                    }
                    else if(tmsg.data[0]==85 && tmsg.data[1]==85
                      && tmsg.data[2]==85 && tmsg.data[3]==85)
                    {
                        led3=!led3;wait(0.1);led3=!led3;wait(0.1)
                          ;
                        led3=!led3;wait(0.1);led3=!led3;
                        systemrun(&tmsg.data[0]);
                    }
                    else//GPS fixed
                    {
                        systemrun(&tmsg.data[0]);
                    }
                }
            }
            else
            {
                tmout_UTC.attach(&atmout_UTC,1.5);
                while(!canpic.read(tmsg) && UTC_wait);
                if(UTC_wait==0) //UTC wait time arrives
                {
```

```
                    UTC_wait=1;
                    led4=1;wait(0.2);led4=0;
                }
            }
        }
}
void systemrun(unsigned char *p_msg)
{
        unsigned char XTend_cmd=0,XTend_nodata=0;
        xtend.send(0x10,0x04,p_msg);
        wait(0.05);//wait 20ms for GPS and AHRS info ready
        t.stop();pc.printf("%f \n\r", t.read());t.reset();
        t.reset(); t.start();//Time logger
        can_send(1,RTR_id[RTR_cmd_1],8,CAN_msg);
        can_rec(RTR_NO[RTR_cmd_1],CAN_data);
        can_send(1,RTR_id[RTR_cmd_4],8,CAN_msg);
        can_rec(RTR_NO[RTR_cmd_4],CAN_data);
        can_send(1,RTR_id[RTR_cmd_6],8,CAN_msg);
        can_rec(RTR_NO[RTR_cmd_6],CAN_data);
        can_send(1,4,8,CAN_msg); can_rec(1,CAN_data);
        can_send(1,7,8,CAN_msg); can_rec(1,CAN_data);
        Airmar_inquire();//Added to inquire info from the Airmar
            PB200
        t.stop();pc.printf("%f \n\r", t.read());t.reset();
        t.reset(); t.start();
        switch(xtend.receive(XTend_cmd,XTend_nodata))
```

```
{
    case 0:     //checksum check fails
        led2=!led2;
        XTend_cmd=0x00;
        pc.printf("00 %d %d \n\r",XTend_cmd,XTend_nodata);
        pc.printf("%d %d %d %d %d %d %d %d %d %d\n\r",xtend.XTend_rec[0],xtend.XTend_rec[1],xtend.XTend_rec[2],xtend.XTend_rec[3],xtend.XTend_rec[4],xtend.XTend_rec[5],xtend.XTend_rec[6],xtend.XTend_rec[7],xtend.XTend_rec[8],xtend.XTend_rec[9]);
        Run_Command(0x33,0);
        XTend_cmd=0;XTend_nodata=0;
        break;
    case 250:   //runstop command
        pc.printf("250 %d %d \n\r",XTend_cmd,XTend_nodata);
        xtend.runstart=0;
        xtend.interrupt(1);
        break;
    case 255:   //timeout-no respond within the timeout
        led2=!led2;
        XTend_cmd=0x00;
        pc.printf("%d %d %d %d %d %d %d %d %d %d\n\r",
            xtend.XTend_rec[0],xtend.XTend_rec[1],xtend.
```

```
            XTend_rec[2],xtend.XTend_rec[3],xtend.
            XTend_rec[4],xtend.XTend_rec[5],xtend.
            XTend_rec[6],xtend.XTend_rec[7],xtend.
            XTend_rec[8],xtend.XTend_rec[9]);
        Run_Command(0x40,0);//previous value 0x45 changed
            on Aug 9
        pc.printf("255 %d %d \n\r",XTend_cmd,XTend_nodata
            );
        XTend_cmd=0;XTend_nodata=0;
        break;
    default:    //data received and checksum check passes
        led3=!led3;
        Run_Command(XTend_cmd,XTend_nodata);
        pc.printf("default %d %d \n\r",XTend_cmd,
            XTend_nodata);
        XTend_cmd=0;XTend_nodata=0;
        break;
  }
  t.stop();printf("%f \n\r", t.read());t.reset();
  xtend.flushserialbuffer();
  GPS_AHRS_on=1;
  motor_on=1;
}
//XTend interrupt function XTend_interrupt
void XTend_interrupt(void)
{
```

```
    unsigned char cmd=0,nodata=0;
    if(xtend.receive(cmd,nodata)==10)
    {
        xtend.runstart=1;
    }
}
//CAN function can_receive
char can_rec(unsigned char counter, unsigned char data[])
{
    float temp1,temp2;
    CANMessage msg;
    char i,ii;
    unsigned char *p=p_msg;         //pointer initial value
    unsigned char *pm=p_motormsg; //pointer to motor message
    while(counter)
    {
        while(!canpic.read(msg));
        counter--;
        if(msg.id < 0x0A)
        {
            switch(msg.id)
            {
                case 2://Error message from the Left motor
                    pc.printf("eL\n\r");
                    led1=!led1;
                    for(ii=0;ii<4;ii++)
```

```
                {
                    *(pm+ii)=*(p_motorfail+ii);
                }
                break;
            case 3://Error message from the Right motor
                pc.printf("eR\n\r");
                pm=p_motormsg+4;
                led1=!led1;
                for(ii=0;ii<4;ii++)
                {
                    *(pm+ii)=*(p_motorfail+ii+4);
                }
                break;
            case 5:
                pm=p_motormsg;
                pc.printf("L\n\r");
                led1=!led1;
                for(ii=0;ii<4;ii++)
                {
                    *(pm+ii)=msg.data[ii];
                }
                break;
            case 6:
                pm=p_motormsg+4;
                pc.printf("R\n\r");
                led1=!led1;
```

```
                    for(ii=0;ii<4;ii++)
                    {
                        *(pm+ii)=msg.data[ii];
                    }
                    break;
                case 9://reserved
                    pc.printf("%x",msg.data[0]);
                    break;
                case 10://reserved
                    break;
                default:
                    break;
            }
        }
        else if(msg.id < 0x20)
        {
            IEEE754_htof(msg.data[0],msg.data[1],msg.data[2],
                msg.data[3],temp1);
            IEEE754_htof(msg.data[4],msg.data[5],msg.data[6],
                msg.data[7],temp2);
            switch(msg.id)
            {
                case 16://Longitude+Latitude
                    break;
                case 17://SOG+COG
                    p=p_msg+8;
```

```
            break;
        case 18://Accelx+Accely
            p=p_msg+16;
            break;
        case 19://Accelz+Angx
            p=p_msg+24;
            break;
        case 20://Angy+Angz
            p=p_msg+32;
            break;
        case 21://MagX+MagY
            p=p_msg+40;
            break;
        case 22://MagZ+M1,1
            p=p_msg+48;
            break;
        case 23://M1,2+M1,3
            p=p_msg+56;
            break;
        case 24://M2,1+M2,2
            p=p_msg+64;
            break;
        case 25://M2,3+M3,1
            p=p_msg+72;
            break;
        case 26://M3,2+M3,3
```

```
                    p=p_msg+80;
                    break;
                case 27://
                    break;
                default:
                    break;
            }
            led1 = !led1;
            for(i=0;i<8;i++)
            {
                *(p+i)=msg.data[i]; //Data recorded
            }
        }
        else
            return 0;
    }
    return 1;
}

//CAN function can_send (RTR or normal message)
void can_send(char RTR_choose,int id, int num, char *pointer)
{
    if(RTR_choose==1)   //RTR message
    {
        if(canpic.write(CANMessage(id,0,num,CANRemote,
            CANStandard)))
```

```
            {
                led1 = !led1 ;
            }
        }
        else                        //Data message
        {
            if(canpic.write(CANMessage(id, pointer, num)))
            {
                led1 = !led1 ;
            }
        }
}
//      CAN acceptance filter configuration
void init_AF(void)
{
        uint32_t address=0;
        //Off mode
        LPC_CANAF->AFMR = 0x00000001;
        //Set explicit standard Frame
        LPC_CANAF->SFF_sa = address;
        //reserved msg and time reference message(id=0 and id=1)
        *((volatile uint32_t *)(LPC_CANAF_RAM_BASE + address)) = (0X001 << 29) | (0X000 << 16) | (0X001 << 13) | (0X001 << 0); address+=4;
        //Error message from Left and Right motor(id=2 and id=3)
```

```
*((volatile uint32_t *)(LPC_CANAF_RAM_BASE + address)) = (0X001 << 29) | (0X002 << 16) | (0X001 << 13) | (0X003 << 0); address+=4;
//RTR Response Data frame from Left & Right motor(id=5 and id=6)
*((volatile uint32_t *)(LPC_CANAF_RAM_BASE + address)) = (0X001 << 29) | (0X005 << 16) | (0X001 << 13) | (0X006 << 0); address+=4;
//Reserved for other usage(id=9 and id=10)
*((volatile uint32_t *)(LPC_CANAF_RAM_BASE + address)) = (0X001 << 29) | (0X009 << 16) | (0X001 << 13) | (0X00A << 0); address+=4;
//Issue the problem when GPS or AHRS lose the connection(id=26 and id=27)
*((volatile uint32_t *)(LPC_CANAF_RAM_BASE + address)) = (0X001 << 29) | (0X01A << 16) | (0X001 << 13) | (0X01B << 0); address+=4;
//Set group standard Frame(id=15~id=25)
LPC_CANAF->SFF_GRP_sa = 0x014;
*((volatile uint32_t *)(LPC_CANAF_RAM_BASE + address)) = (0X001 << 29) | (0X00f << 16) | (0X001 << 13) | (0X019 << 0); address+=4;
//Set explicit extended Frame for CAN 1
LPC_CANAF->EFF_sa = 0x018;
*((volatile uint32_t *)(LPC_CANAF_RAM_BASE + address)) = (0X000 << 29) | (0X9f11223); address+=4;  //127250
```

```
message
*((volatile uint32_t *)(LPC_CANAF_RAM_BASE + address)) = (0X000 << 29) | (0X9f11323); address+=4; //127251 message
*((volatile uint32_t *)(LPC_CANAF_RAM_BASE + address)) = (0X000 << 29) | (0X9f80123); address+=4; //129025 message
*((volatile uint32_t *)(LPC_CANAF_RAM_BASE + address)) = (0X000 << 29) | (0X9f80223); address+=4; //129026 message
*((volatile uint32_t *)(LPC_CANAF_RAM_BASE + address)) = (0X000 << 29) | (0X9fd0223); address+=4; //130306 message
*((volatile uint32_t *)(LPC_CANAF_RAM_BASE + address)) = (0X000 << 29) | (0Xdf11923); address+=4; //127257 message
*((volatile uint32_t *)(LPC_CANAF_RAM_BASE + address)) = (0X000 << 29) | (0Xdf80923); address+=4; //129033 message
*((volatile uint32_t *)(LPC_CANAF_RAM_BASE + address)) = (0X000 << 29) | (0X15fd0623);address+=4; //130310 message
*((volatile uint32_t *)(LPC_CANAF_RAM_BASE + address)) = (0X000 << 29) | (0X15fd0723);address+=4; //130311 message
// Set group extended Frame
```

```
    LPC_CANAF->EFF_GRP_sa = 0x03C;
    // Set End of Table
    LPC_CANAF->ENDofTable = 0x03C;
    //normal mode
    LPC_CANAF->AFMR = 0x00000000;
}
// Transform the information from byte to float
void IEEE754_htof(unsigned char a,unsigned char b,unsigned
    char c,unsigned char d, float& val)
{
    long temp=0;
    temp|=a;temp<<=8;temp|=b;temp<<=8;
    temp|=c;temp<<=8;temp|=d;
    float *p=(float *)&temp;
    val=*p;
}
//This function is used to transform the float data to byte
    data for transmission on the CAN bus
void IEEE754_ftoh(float val,unsigned char& t1,unsigned char&
    t2,unsigned char& t3,unsigned char& t4)
{
    long *p=(long *)&val;
    long temp=*p;
    t4=temp&0xff;
    temp>>=8; t3=temp&0xff;
    temp>>=8; t2=temp&0xff;
```

```
    temp>>=8; t1=temp&0xff;
}
```

A.2 The navigation CAN node program

The navigation CAN node program is provided in the following part.

```
#include "mbed.h"
#include "GPS.h"
#include "AHRS.h"
#include "math.h"
#include "Func_init.h"
//Initialize the LEDs
DigitalOut led1(LED1);  //CAN sent successfully(blink)
DigitalOut led2(LED2);  //GPS data invalid(blink)
DigitalOut led3(LED3);  //AHRS data invalid(blink)
DigitalOut led4(LED4);  //Program runs normally(blink)
//Interfaces defination
CAN navigator_can(p30, p29);//rd, td (connected with MCP2551)
Serial pc(USBTX, USBRX);    //tx, rx
GPS gps(p13, p14);          //tx, rx
AHRS ahrs(p9, p10);         //tx, rx
Timer t;
unsigned char msg_send[88]={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};//Rows=11, Columns=8
char *p_msg=(char *)&msg_send[0];    //pointer to the 1st CAN message address
```

```
char can_msg[8]={0x00};
//Main function
int main()
{
    int i;
    led1=0;led2=0;led3=0;led4=0;
    gps.initial();
    navigator_can.frequency(1000000);//CAN freq configured as
        1MHz (CAN frequency 125000bps)
    init_AF();  //CAN filter configuration
                //Only accept the id=9 and id=10 message
    navigator_can.attach(&can_interrupt);
    gps.sample();
    while(1)
    {
        switch(gps.sample())
        {
            case 0://data not fixed
                led2=1;wait(0.1);led2=0;      //LED2 blink
                for(i=0;i<gps.number+1;i++)
                {
                    pc.printf("%c",gps.msg[i]);
                }
                pc.printf("\r\n");
                break;
            case 1://data valid
```

```
            pc.printf("1 \r\n");
            for(i=0;i<gps.number+1;i++)
            {
                pc.printf("%c",gps.msg[i]);
            }
            pc.printf("\r\n");
            IEEE754_ftoh(gps.longitude,msg_send[0],msg_send[1],msg_send[2],msg_send[3]); //longitude
            IEEE754_ftoh(gps.latitude ,msg_send[4],msg_send[5],msg_send[6],msg_send[7]); //latitude
            IEEE754_ftoh(gps.sog  ,msg_send[8],msg_send[9],msg_send[10],msg_send[11]);         // speed over ground (SOG)
            IEEE754_ftoh(gps.cog  ,msg_send[12],msg_send[13],msg_send[14],msg_send[15]);       // course over ground (COG)
            break;
        case 2://No gps signal at all
            for(i=0;i<gps.number+1;i++)
            {
                pc.printf("%c",gps.msg[i]);
            }
            pc.printf("\r\n");
            break;
```

```
            case 255://Checksum fails
                pc.printf("255 \r\n");
                break;
            default:
                pc.printf("default");
                break;
        }
        if(ahrs.sample(0xcc,79))
        {
            for(i=0;i<72;i++)
            {
                msg_send[16+i]=ahrs.rec[i+1];
            }
        }
        else
        {
            led3=1;wait(0.1);led3=0;
        }
        led4=!led4;//indicate that the program is running
    }
}
void can_interrupt(void)
{
    CANMessage msg;
    //Check if CAN message received
    if(navigator_can.read(msg))
```

```
    {
        if(msg.type==CANRemote)//RTR message
        {
            switch(msg.id)
            {
                case 16://GPS data-longitude and latitude
                    can_send(0x010,8,p_msg);
                    break;
                case 17://GPS data-longitude latitude and SOG
                    and COG
                    can_send(0x010,8,p_msg);
                    can_send(0x011,8,p_msg+8);
                    break;
                case 18://Accel x, y and z, and AngRate x
                    can_send(0x012,8,p_msg+16);
                    can_send(0x013,8,p_msg+24);
                    break;
                case 19://Accel z and AngRate x, y and z
                    can_send(0x013,8,p_msg+24);
                    can_send(0x014,8,p_msg+32);
                    break;
                case 20://Accel x, y and z; AngRate x, y and
                  z
                    can_send(0x012,8,p_msg+16);
                    can_send(0x013,8,p_msg+24);//wait(0.07)
                        ;//?time too long?
```

```
            can_send(0x014,8,p_msg+32);
            break;
        case 21://GPS info; Accel x, y and z; AngRate x, y and z
            can_send(0x010,8,p_msg);
            can_send(0x011,8,p_msg+8);
            can_send(0x012,8,p_msg+16);wait(0.07);
            can_send(0x013,8,p_msg+24);
            can_send(0x014,8,p_msg+32);
            break;
        case 22://AHRS MagX MagY MagZ M1.1
            can_send(0x015,8,p_msg+40);
            can_send(0x016,8,p_msg+48);
            break;
        case 23://MagZ and Rotation Matrix
            can_send(0x016,8,p_msg+48);
            can_send(0x017,8,p_msg+56);
            can_send(0x018,8,p_msg+64);wait(0.07);
            can_send(0x019,8,p_msg+72);
            can_send(0x01A,8,p_msg+80);
            break;
        case 26:
            led4=1;wait(0.1);led4=0;
            break;
        case 28:
            break;
```

```
                    default:
                        break;
                }
            }
            else//data message
            {
                switch(msg.id)
                {
                    case 15:
                        break;
                }
            }
        }
}
void init_AF(void)
{
        uint32_t address=0;
        //Off mode
        LPC_CANAF->AFMR = 0x00000001;
        //Set explicit standard Frame
        LPC_CANAF->SFF_sa = address;
        //Reserved for other usage(id=9 and id=10)
        *((volatile uint32_t *)(LPC_CANAF_RAM_BASE + address)) =
            (0X001 << 29) | (0X009 << 16) | (0X001 << 13) | (0X00A
            << 0); address+=4;
        //Set group standard Frame
```

```
    LPC_CANAF->SFF_GRP_sa = 0x004;
    //(id=15~id=28)
    *((volatile uint32_t *)(LPC_CANAF_RAM_BASE + address)) = (0X001 << 29) | (0X00f << 16) | (0X001 << 13) | (0X01C << 0);
    //Set explicit extended Frame
    LPC_CANAF->EFF_sa = 0x008;
    // Set group extended Frame
    LPC_CANAF->EFF_GRP_sa = 0x008;
    // Set End of Table
    LPC_CANAF->ENDofTable = 0x008;
    //normal mode
    LPC_CANAF->AFMR = 0x00000000;
}
//This function if used for CAN message sending
void can_send(int id, int num, char *pointer)
{
    if(navigator_can.write(CANMessage(id, pointer, num)))
    {
        led1=!led1;//CAN message sent successfully
    }
}
//This function is used to transform the float data to byte data for transmission on the CAN bus
void IEEE754_ftoh(float val,unsigned char& t1,unsigned char& t2,unsigned char& t3,unsigned char& t4)
```

```
{
    long *p=(long *)&val;
    long temp=*p;
    t4=temp&0xff;
    temp>>=8; t3=temp&0xff;
    temp>>=8; t2=temp&0xff;
    temp>>=8; t1=temp&0xff;
}
//This function is used to transform the information from
    byte to float for calculation
void IEEE754_htof(unsigned char a,unsigned char b,unsigned
    char c,unsigned char d, float& val)
{
    long temp=0;
    temp|=a;temp<<=8;temp|=b;temp<<=8;
    temp|=c;temp<<=8;temp|=d;
    float *p=(float *)&temp;
    val=*p;
}
```

A.3 The motor controller CAN node program

The motor controller CAN node program is provided in the following part.

```
#include <p18f258.h>      // PIC Controller header file
#include <usart.h>
#include <delays.h>
```

```
#include <timers.h>
#include "datatype.h"
#include "functions.h"
//Function declaration
void rx_handler (void);
//Global variables declaration
uint8 rs485_msg[15];
uint8 rs485_r[13]={0x01,0x02,0x03,0x04};
uint8 rs485_updt[8]={0x01,0x00,0x00,0x32};
uint8 rs485_status;
//Macro define
#define L_or_R 0
//Necessary configuration for PIC
#pragma config WDT=OFF  //Disable watchdog timer
#pragma config OSC=HS   //Oscillator selection
#pragma config OSCS=OFF
#pragma config LVP=OFF
#pragma code rx_interrupt = 0x8
void rx_int (void)
{
  _asm goto rx_handler _endasm
}
#pragma code
#pragma interrupt rx_handler
void rx_handler (void)
{
```

```
INTCONbits.GIE=0;
if(RXB0CONbits.RXRTRRO)
{
  #if L_or_R==1
  {
    if(RXB0SIDL==0x90)
    {
      if(!rs485_status)
        can_send(0x0040,8,rs485_updt);
      else
        can_send(0x00a0,8,rs485_updt);
    }
  }
  #else
  {
    if(RXB0SIDL==0xf0)
    {
      if(!rs485_status)
        can_send(0x0060,8,rs485_updt);
      else
        can_send(0x00c0,8,rs485_updt);
    }
  }
  #endif
}
else
```

```
{
  #if L_or_R==1
  {
    rs485_updt[0]=RXB0D0;
    if(rs485_updt[0]>0x20) rs485_updt[0]=0x20;
    rs485_updt[1]=RXB0D1; rs485_updt[2]=RXB0D2; rs485_updt
        [3]=RXB0D3;
  }
  #else
  {
    rs485_updt[0]=RXB0D4;
    if(rs485_updt[0]>0x20) rs485 _updt[0]=0x20;
    rs485_updt[1]=RXB0D5; rs485_updt[2]=RXB0D6; rs485_updt
        [3]=RXB0D7;
  }
  #endif
}
PIR3bits.RXB0IF=0;
RXB0CONbits.RXFUL=0;
INTCONbits.GIE=1;
}
//main function
void main(void)
{
  uint8 i;
  INTCON = 0x00;            //disable all interrupts
```

```
//Initialization of all
pin_init();
usart_init();
can_init();
timer0_init();
//motor initialization command
msg_switch(0);
rs485_send(15,rs485_msg);
//Enable global interrupt enable bit
INTCON=0xc0;  //enable interrupt
while(1)
{
  //check received data status of motor
  if(rs485_rev(9))  //if reception data is received successfully
  {
    rs485_status=1; //rs485 connection right
  }
  else
  {
    rs485_status=0; //rs485 connection fail
  }
  //set command for motor
  msg_switch(1);
  rs485_send(15,rs485_msg);
}
```

```
}
//Initialization of all modules for PIC18f258
void pin_init(void)
{
  //Microcontroller Pin Initialization
  PORTA=0;TRISA=0;
  PORTB=0;TRISB=0;
  PORTC=0;TRISC=0;
}
//Initialization of UART for PIC18f258
void usart_init(void)
{
  TRISCbits.TRISC6=0;//Define RX as input
  TRISCbits.TRISC7=1;//Define TX as output
  //Open USART configured as 8-bit data, 9600 baud
  //Include the config of TXEN and SPEN enable
  //and USART pin RC6/TX and RC7/RX config
  OpenUSART ( USART_TX_INT_OFF &
              USART_RX_INT_OFF &
              USART_ASYNCH_MODE &
              USART_EIGHT_BIT &
              USART_CONT_RX &
    USART_BRGH_HIGH, 129);
  delayms(100);
}
//Initialization of timer0 module for PIC18f258
```

```
void timer0_init(void)
{
  //1:256 prescale value, 16 bit timer
  T0CON = 0x07;          // Configure timer, but don't start it yet
     TMR0H = 0x67;              // Reset Timer0 to 0x6769—follow the steps first-H, then-L
     TMR0L = 0x69;       // 2s timer(1s=0xB3B5)
     INTCONbits.TMR0IF = 0;   // Clear Timer0 overflow flag
}
//Initialization of CAN module for PIC18f258
void can_init(void)
{
//Pin config-RB3/CANRX, RB2/CANTX
  TRISBbits.TRISB3=1;
  TRISBbits.TRISB2=0;
//Configuration mode-wait
  CANCON=0x80;
  while(~CANSTATbits.OPMODE2);
     BRGCON1=0x00; //SJW=1*TQ;TQ=(2*1)/20Mbps;TQ=0.1us;
     BRGCON2=0x98; //Prop=1*TQ; Phase1=4*TQ
     BRGCON3=0x03; //Phase2=4*TQ
  TXB0CON=0X03; //Transmit priority bits (buffer priority)
             //highest priority
  TXB0SIDH=0x00; //id=0b->00000100000
  TXB0SIDL=0x20;
```

```
  TXB0SIDLbits.EXIDE=0;//standard identifier 11 bits
  TXB0DLC=0X08; //Data length
  //Data_Send
  TXB0D0=0xff;
  TXB0D1=0xff;
  TXB0D2=0xff;
  TXB0D3=0xff;
  TXB0D4=0xff;
  TXB0D5=0xff;
  TXB0D6=0xff;
  TXB0D7=0xff;
//Receive registor 0 configuration
  RXB0CON=0X00; //receive all valid messages
  RXB0SIDH=0X00;
  RXB0SIDL=0X00;
  RXB0DLC=0X08;
  RXB0D0=0X00;
  RXB0D1=0X00;
  RXB0D2=0X00;
  RXB0D3=0X00;
  RXB0D4=0X00;
  RXB0D5=0X00;
  RXB0D6=0X00;
  RXB0D7=0X00;
//Mask and Filter configuration
  RXM0SIDH=0xff;
```

```
    RXM0SIDL=0Xe0;
    #if L_or_R==1
        //Filter config-only accept id=0x0080
        RXF0SIDH=0x00;
        RXF0SIDL=0x80;
        RXF0SIDLbits.EXIDEN=0;
    #else
        //Filter config-only accept id=0x00e0
        RXF0SIDH=0x00;
        RXF0SIDL=0xe0;
        //RXF0SIDL=0x80;
        RXF0SIDLbits.EXIDEN=0;
    #endif
//Normal mode-wait
    CANCON=0x00;
    while(CANSTATbits.OPMODE2);
//Initialize the CAN interrupt
    PIR3=0x00;          //clear all interrupt flag
    PIE3=0x01;
    IPR3=0x01;
}
void can_send(uint16 id, uint8 num, uint8 msg[])
{
    TXB0SIDH=(id>>8)&0xff;   //id_H
    TXB0SIDL=id&0xff;    //id_L
    TXB0SIDLbits.EXIDE=0;//standard identifier 11 bits
```

```
  //Data length
  TXB0DLC=num;
  //Data_Send
  TXB0D0=msg[0];
  TXB0D1=msg[1];
  TXB0D2=msg[2];
  TXB0D3=msg[3];
  TXB0D4=msg[4];
  TXB0D5=msg[5];
  TXB0D6=msg[6];
  TXB0D7=msg[7];
  TXB0CONbits.TXREQ=1;
  while(~PIR3bits.TXB0IF);
  TXB0CONbits.TXREQ=0;
}
void msg_switch(uint8 sw)
{
  switch(sw)
  {
    case 0://initialize the motor
      config_msg_motor(0x00,0x00,0x00,0x00);
      break;
    case 1://set command
      config_msg_motor(rs485_updt[0],rs485_updt[1],rs485_updt
         [2],rs485_updt[3]);//0x02,0x00,0x01,0x64
      break;
```

```
    case 2://query command
      get_msg_motor();
      break;
    //case 3://CAN message
    //  config_msg_motor(rs485_updt[0],rs485_updt[1],
        rs485_updt[2],rs485_updt[3]);
    //  break;
    default:
      break;
  }
}
uint8 config_msg_motor(uint8 spd_h,uint8 spd_l,uint8
    direction ,uint8 power)
{
  uint8 i,status;
  uint16 crc;
  rs485_msg[0] = 0x80;            // Destination Address
  rs485_msg[1] = 0x10;            // Source Address
  rs485_msg[2] = 0x01;            // PCB
  rs485_msg[3] = 0x10;            // INS
  rs485_msg[4] = 0x00;            // ID MSB
  rs485_msg[5] = 0x12;            // ID LSB ?? 0x12 in the
    UAV project
  rs485_msg[6] = 0x04;            // Length of Data
  rs485_msg[7] = spd_h;           // Speed-RPM-MSB
  rs485_msg[8] = spd_l;           // Speed-RPM-LSB
```

```
  rs485_msg[9] = direction;         // Direction
  rs485_msg[10] = power;            // Power (0 - 100% -> 0x00 - 0x64)
  crc=cal_crc(0,rs485_msg,11);
  rs485_msg[12] = crc^0xff;    // crc_low
  rs485_msg[11] = (crc>>8)^0xff;   // crc_high
  rs485_msg[13] = 0xff;         // dummy byte for rs485-driver dir.
  rs485_msg[14] = 0xff;       // dummy byte for rs485-driver dir.
  return 1;
}
uint8 get_msg_motor(void)          //len=7;
{
  uint16 crc;
  //Query Software from Torqeedo thruster
  rs485_msg[0] = 0x80;              // Destination Address
  rs485_msg[1] = 0x10;              // Source Address
  rs485_msg[2] = 0x01;              // PCB
  rs485_msg[3] = 0x20;              // INS-get
  rs485_msg[4] = 0x00;              // ID MSB
  rs485_msg[5] = 0x01;              // ID LSB
  //rs485_msg[5] = 0x50;              // ID LSB supply voltage check
  rs485_msg[6] = 0x00;              // Length of Data
```

```
  rs485_msg[9] = 0xff;        // dummy byte for rs485-driver dir.
  rs485_msg[10] = 0xff;       // dummy byte for rs485-driver dir.
  crc=cal_crc(0,rs485_msg,7);
  rs485_msg[8] = crc^0xff;     //crc_low-CHK0
  rs485_msg[7] = (crc>>8)^0xff; //crc_high-CHK1
  return 1;
}
//rs485_send
uint8 rs485_send(uint8 num, uint8 msg[])
{
  uint8 i;
  TRISAbits.TRISA0=1;//SP485_TX_EN
  delayms(10);//necessary delay
        //This delay solve the problem of the information initial diffe
        //rence between the two SBC28PCs
  for(i=0;i<num;i++)
  {
    while(BusyUSART());
    WriteUSART(msg[i]);
  }
  //delayms(10);
  while(BusyUSART());//delayms(1);
  TRISAbits.TRISA0=0;//SP485_RX_EN
```

```
  return 1;
}
//rs485_receive
uint8 rs485_rev(uint8 num)
{
  uint8 i;
  T0CONbits.TMR0ON = 1;//Start Timer 0
                  //2s idle->break out
  for(i=0;i<num;i++)
  {
    while(!PIR1bits.RCIF)
    {
      if(INTCONbits.TMR0IF)
      {
        timer0_init();
        return 0;
      }
    }
    rs485_r[i]=RCREG;
  }
  timer0_init();
  return 1;
}
uint16 cal_crc(uint16 crc, uint8 *ptr, uint16 len)
{
```

```
  static const uint8 oddparity[16] = { 0, 1, 1, 0, 1, 0, 0,
     1, 1, 0, 0, 1, 0, 1, 1, 0 };
    uint16 idata;

    for(; len; --len)
    {
        idata = (*ptr ^ crc) & 0xff;
        ptr++;
        crc >>= 8;
        if(oddparity[idata & 0x0f] ^ oddparity[idata >> 4])
          crc ^= 0xc001;
        idata <<= 6;
        crc ^= idata;
        idata <<= 1;
        crc ^= idata;
    }
    return crc;
}
//delay (1~65535)ms
void delayms(uint16 tm)
{
    do
  {
    Delay100TCYx(50); //1ms
  }while(--tm);
}
```

```
//delay (1~255)s
void delays(uint8 tm)
{
	do
	{
	Delay10KTCYx(250);//500ms
	Delay10KTCYx(250);//500ms
    }while(--tm);
}
```