Design and Implementation of a Glider Control System

HANNAH LINDBERG
Design and Implementation of a Glider Control System

HANNAH LINDBERG

Master's Thesis in Systems Engineering (30 ECTS credits)
Master Programme in Aerospace Engineering (120 credits)
Royal Institute of Technology year 2015
Supervisor at Airbus Defence & Space: Dr. Mädiger and Bernd Langpap
Supervisor at KTH was Per Engvist
Examiner was Per Engvist

TRITA-MAT-E 2015:16
ISRN-KTH/MAT/E--15/16--SE

Royal Institute of Technology
School of Engineering Sciences

KTH SCI
SE-100 44 Stockholm, Sweden

URL: www.kth.se/sci
Acknowledgements

First, I would like to thank Dr. Mädgier for giving me the opportunity to work on this project and for his support and guidance. I would also like to thank Bernd Langpap for exceptional feedback, guidance and dedication. Both Dr. Mädgier and Mr. Langpap have been wonderful supervisors at Airbus Defence and Space. Furthermore, I would also like to thank my supervisor at KTH, Dr. Enqvist, for excellent orientation, administrative organisation and continuous feedback on my report.

For all shared ideas, discussions and involvement that has given me a broader understanding of the project as a whole but also of this field of engineering I would like to thank Dr. Wilde, Uwe Soppa, Isabel Bustos and Ingo Möller.

Last, to Stephan Von-Deetzen, Verena Luebke, Ralf Regele and Dr. Nuber for being helpful and cheerful colleagues working beside me in the space robotics laboratory.

Bremen, May 11, 2015

Hannah Lindberg
Abstract

ROBEX is a unique research project combining Airbus Defence and Space’s robotics expertise with deep-sea exploration technology to discover more about the most extreme environments known to man. As a part of this project, a deep-sea glider called MOTH, is under development with the objective to determine whether gliders can be used as a platform for bathymetric and electromagnetic soundings of the seafloor as well as for new water column research. This master’s thesis aims to design and implement the MOTH glider’s control system.

The glider will have an independent emergency system, a power unit, an on-board computer (OBC), actuators, navigation sensors and scientific measurement instruments which can be swapped between missions and are connected via remote terminal units. The selected OBC is a Linux embedded Axotec GX-6300 with RS232 and CAN bus interfaces, as selected in the electrical architecture, and the chosen operating system is Linux Debian. The glider communicates with GNS/Iridium antenna and also has an ethernet cable link for ground station operations and a future option of an acoustic transceiver.

To control actuation, the glider is equipped with a rudder, a left and a right wing flap, a moveable mass and a buoyancy tank. It travels in sawtooth patterns and is therefore always descending, ascending or transitioning during operation and at times ascending all the way to the surface to transmit and receive data via satellite communication. A model-based feedback controller for longitudinal control has been programmed based on the equations of motion described in this report. The modelled longitudinal trajectory is as desired until a transition point is reached, the model is, presumable because of the uncertainty of the model parameters, unstable as the actuators are unable to correct the pitch angle.

An AHRS navigation sensor emulator and an OBC emulator have been programmed to simulate the communication between these two and the emulated system is well operating both as a continuous stream and for polling data. The emulator and the pitch controller, when updated parameter values are available, will be used for simulation and verification tests in the laboratory environment.

The ROBEX alliance will, if the objectives with the MOTH glider are met, continue to design gliders with the aim to increase the maximum duration time and speed in order to reach greater depths of the oceans.

Keywords: ROBEX, MOTH, glider, buoyancy, OBC, actuators, pitch, deflection, AHRS

Note: This thesis has under agreement with Airbus DS been allowed publishing without any confidentiality restrictions.
# Contents

1 Symbols and Acronyms ................................................. 7

2 Introduction .......................................................... 9

3 Problem
   3.1 Objectives .................................................................. 11
   3.2 Problem Description .................................................. 12
   3.3 Approach .................................................................. 13

4 System Description ....................................................... 13
   4.1 System Architecture .................................................. 14
   4.2 Glider Behaviour ...................................................... 15
   4.3 Glider Dynamics ....................................................... 16

5 Electrical System Architecture ....................................... 21
   5.1 CAN bus, RS232 & Ethernet Cable link .......................... 23
   5.2 Sensor & Actuator Data .............................................. 23
   5.3 Science Data & RTUs ............................................... 24
   5.4 Telemetry & Telecommunication .................................. 24
   5.5 On-Board Computer ................................................ 24
   5.6 Operating System ................................................... 24
   5.7 Solid State Memory .................................................. 25

6 System State Modelling .................................................. 26
   6.1 Design .................................................................. 26
   6.2 State Control Signalling ............................................. 28

7 Pitch Control
   7.1 Equations for Longitudinal Control .............................. 30
   7.2 Model-Based Feedback Control ................................... 31
   7.3 Simulink block diagram .......................................... 33
   7.4 Results ............................................................... 35

8 Software Architecture ................................................... 39

9 AHRS Sensor Emulator .................................................. 40
   9.1 AHRS ................................................................. 40
   9.2 Data Communication Protocol .................................... 42
## 9.3 AHRS and OBC emulators
9.3.1 Continuous data stream programme
9.3.2 Polling data programme

### 10 Conclusion

### 11 Discussion

### 12 Appendices

<table>
<thead>
<tr>
<th>Appendix</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Functional Requirements</td>
<td>53</td>
</tr>
<tr>
<td>B</td>
<td>Device list</td>
<td>57</td>
</tr>
<tr>
<td>C</td>
<td>Glider Dimensions</td>
<td>58</td>
</tr>
<tr>
<td>D</td>
<td>OBC Trade-off</td>
<td>60</td>
</tr>
<tr>
<td>E</td>
<td>OS Trade-off</td>
<td>61</td>
</tr>
</tbody>
</table>
List of Figures

2.1 ROBEX MOTH Deep-Sea Glider ........................................... 11
3.1 The V-model ................................................................. 12
3.2 System level breakdown with thesis tasks marked in yellow and blue ........................................ 13
4.1 Glider architecture overview ............................................. 14
4.2 OBC task overview .......................................................... 15
4.3 System level use case ....................................................... 16
4.4 OBC use case ................................................................. 17
4.5 Wing profile, glider coordinate system ............................... 18
4.6 Coordinate system for longitudinal control calculations .......... 19
4.7 General closed-loop controller ........................................... 21
5.1 Electrical System Architecture ........................................... 22
5.2 Example of preemptive scheduling ..................................... 25
6.1 MOTH System State Machine ............................................ 29
7.1 Glider mass definitions ..................................................... 30
7.2 Glider trajectory with altitude boundaries ......................... 33
7.3 Matlab Simulink block-diagram of the designed closed loop feedback controller ................. 34
7.4 Glider displacement in x- resp. z-direction .......................... 35
7.5 Glider displacement in x-z plane ........................................ 36
7.6 Pitch angle and flap deflection .......................................... 36
7.7 Pitch error ..................................................................... 37
7.8 Free stream velocity, flight path angle and angle of attack ...... 37
7.9 Actuator torques ............................................................... 38
7.10 Angular velocity .............................................................. 38
8.1 Flight hardware implementation of application and basic S/W ........ 39
8.2 Closed loop simulator implementation of application and basic S/W ........................................... 40
9.1 Emulated system overview on the left, real operational system to the right ......................... 41
9.2 3DM-GX4-25 dimensions and connector .............................. 41
9.3 Communication diagram between AHRS and OBC .................. 43
9.4 Header & Checksum Overview ............................................ 44
9.5 Packet Payload Overview .................................................. 44
9.6 Set to Idle command: 7565 0102 0202 E1C7 ......................... 44
9.7 OBC Emulator Interface ..................................................... 49
**List of Tables**

4.1 Glider Actuator Concept ................................................................. 21
6.1 List of States ...................................................................................... 26
6.2 List of Events ...................................................................................... 27
7.1 Parameters .......................................................................................... 32
7.2 Simulink block inputs and outputs ....................................................... 34
9.1 3DM-GX4-25 technical specification .................................................... 42
9.2 Command and data overview ................................................................. 43
9.3 Hexadecimal code compared with decimal code ..................................... 45
9.4 AHRS commands and replies for continuous stream .............................. 46
9.5 AHRS commands and replies for polling data ......................................... 46
9.6 AHRS input data generated by 6DOF model simulator ............................ 47
12.1 Glider device list ................................................................................ 57
## 1 Symbols and Acronyms

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>Angle of attack</td>
<td>[$^\circ$]</td>
</tr>
<tr>
<td>$\varphi$</td>
<td>Flight path angle</td>
<td>[$^\circ$]</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Pitch angle</td>
<td>[$^\circ$]</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Sideslip angle</td>
<td>[$^\circ$]</td>
</tr>
<tr>
<td>$\delta$</td>
<td>Deflection angle</td>
<td>[$^\circ$]</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Fluid density</td>
<td>[kg/m³]</td>
</tr>
<tr>
<td>$\omega$</td>
<td>Angular velocity</td>
<td>[rad/s]</td>
</tr>
<tr>
<td>$T$</td>
<td>Torque</td>
<td>[Nm]</td>
</tr>
<tr>
<td>$M$</td>
<td>Moment</td>
<td>[Nm]</td>
</tr>
<tr>
<td>$F$</td>
<td>Force</td>
<td>[N]</td>
</tr>
<tr>
<td>$B$</td>
<td>Buoyancy force</td>
<td>[N]</td>
</tr>
<tr>
<td>$I$</td>
<td>Moment of inertia</td>
<td>[kgm²]</td>
</tr>
<tr>
<td>$E$</td>
<td>Kinetic energy</td>
<td>[J]</td>
</tr>
<tr>
<td>$v$</td>
<td>Velocity</td>
<td>[m/s]</td>
</tr>
<tr>
<td>$V_\infty$</td>
<td>Free stream velocity</td>
<td>[m/s]</td>
</tr>
<tr>
<td>$A$</td>
<td>Area</td>
<td>[m²]</td>
</tr>
<tr>
<td>$x$</td>
<td>Length</td>
<td>[m]</td>
</tr>
<tr>
<td>$y$</td>
<td>Width</td>
<td>[m]</td>
</tr>
<tr>
<td>$m$</td>
<td>mass</td>
<td>[kg]</td>
</tr>
<tr>
<td>$C_D$</td>
<td>Drag coefficient</td>
<td>[ ]</td>
</tr>
<tr>
<td>$C_L$</td>
<td>Lift coefficient</td>
<td>[ ]</td>
</tr>
<tr>
<td>$C_m$</td>
<td>Moment coefficient</td>
<td>[ ]</td>
</tr>
<tr>
<td>$V$</td>
<td>Volume</td>
<td>[m³]</td>
</tr>
<tr>
<td>$r$</td>
<td>Distance to center of buoyancy</td>
<td>[m]</td>
</tr>
<tr>
<td><strong>Acronym</strong></td>
<td><strong>Full form</strong></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td>AHRS</td>
<td>Attitude Heading Reference System</td>
<td></td>
</tr>
<tr>
<td>ASW</td>
<td>Application Software</td>
<td></td>
</tr>
<tr>
<td>AUV</td>
<td>Autonomous underwater vehicle</td>
<td></td>
</tr>
<tr>
<td>BSS</td>
<td>Basic Software Simulator</td>
<td></td>
</tr>
<tr>
<td>BSW</td>
<td>Basic Software</td>
<td></td>
</tr>
<tr>
<td>CB</td>
<td>Center of buoyancy</td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
<td></td>
</tr>
<tr>
<td>CTD-O</td>
<td>Conductivity-Temperature-Depth-Optical</td>
<td></td>
</tr>
<tr>
<td>EF</td>
<td>Estimation filters</td>
<td></td>
</tr>
<tr>
<td>EMR</td>
<td>Emergency Mass Release</td>
<td></td>
</tr>
<tr>
<td>FDIR</td>
<td>Failure, Detection, Isolation and Recovery</td>
<td></td>
</tr>
<tr>
<td>FIFO</td>
<td>First in, first out</td>
<td></td>
</tr>
<tr>
<td>GNC</td>
<td>Guidance Navigation Control</td>
<td></td>
</tr>
<tr>
<td>GNS</td>
<td>Global Navigation Sensor</td>
<td></td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
<td></td>
</tr>
<tr>
<td>IMU</td>
<td>Inertial Measurement Unit</td>
<td></td>
</tr>
<tr>
<td>LLH</td>
<td>Latitude, longitude and Height</td>
<td></td>
</tr>
<tr>
<td>NED</td>
<td>North east down</td>
<td></td>
</tr>
<tr>
<td>MIP</td>
<td>Matlab integration package</td>
<td></td>
</tr>
<tr>
<td>OBC</td>
<td>On-board computer</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
<td></td>
</tr>
<tr>
<td>PCDU</td>
<td>Power Conditioning Distribution Unit</td>
<td></td>
</tr>
<tr>
<td>ROBEX</td>
<td>Robotic Exploration of Extreme Environments</td>
<td></td>
</tr>
<tr>
<td>RTU</td>
<td>Remote Terminal Unit</td>
<td></td>
</tr>
<tr>
<td>S/S</td>
<td>Subsystem</td>
<td></td>
</tr>
<tr>
<td>SOFAR</td>
<td>Sound Fixing and Ranging</td>
<td></td>
</tr>
<tr>
<td>SSM</td>
<td>Solid State Memory</td>
<td></td>
</tr>
<tr>
<td>TM/TC</td>
<td>Telemetry/Telecommunications</td>
<td></td>
</tr>
<tr>
<td>UTC</td>
<td>Universal Time Coordinated</td>
<td></td>
</tr>
</tbody>
</table>
2 Introduction

Oceanography, a Greek composition of the two words "ocean" and "write", is the name of the Earth science branch that studies the ocean. It began already in pre-historic times, as observations on tides were recorded by Aristotle and Strabo and the exploration of the seas and oceans continued primarily for studying currents and for cartography. Despite the early studies, knowledge of the oceans remained confined to the topmost, shallow, layers of the water and very little is to this date known about the bottom layers of the deep ocean.

The study of the oceans is allied to understanding global climate changes, and because of evaporation, precipitation, thermal flux and solar insolation the atmosphere and oceans are bound. Atmospheric carbon dioxide and wind stress driving the ocean currents prevail the ocean’s biogeochemical setup. Even though 71% of Earth’s surface is covered by water more than 95% of it is still unexplored, humans have acquired more knowledge of the surface of the moon than the bottom of our oceans, yet the importance of understanding the oceans and their role in the planet’s ecosystem cannot be overemphasised.

In extreme environments, such as in the deepest points of the ocean, the arctic and antarctic regions or in space, manned systems operate at great expense and risk, leading an increasing desire for autonomous robotic systems. These systems would have been rather ambitious to find in Aristotle’s time and because of the challenges caused by difficult environmental conditions, e.g. cost, limited communication, power budget, salt water, application constraints (e.g. handling by crane in rough water) and limited accommodation space, it still is not a simple development task. All of these constraints lead to the necessity of a strong systems engineering approach, in particular concerning the design of the avionics system. Nevertheless, the autonomous underwater vehicles (AUV) have emerged, from Henry Stommel and John Swallow’s neutrally buoyant float in the mid-1950s that measured currents whilst tracked by a following ship, to the acoustic SOFAR (Sound Fixing and Ranging) float in the 1960’s developed by Doug Webb and Tim Rossby. The SOFAR transmitted sound at predefined intervals which were picked up by autonomous receivers at fixed locations in the ocean [1]. The SOFAR later got upgraded to RAFO$^1$, which had hydrophones on the floats and could thus record times of arrival of the acoustic signals and surface by releasing a weight and communicate their data via satellite.

Then came the bobber float, which could control its buoyancy and depth by cycling up and down in saw-tooth patterns using an electric pump that moved oil between the internal reservoir and an external bladder. In the late 1980’s, Webb developed a thermally powered buoyancy engine which during ascent from cold depths to the relatively warm surface use the temperature change to warm and expand a working fluid. The possibility to perform thousands of vertical cycles came with this new

\(^1\)SOFAR backwards
endurance and led Webb to extend his idea to implement the thermal engine in an underwater glider, together with Stommel, they came to develop the Slocum glider concept. In the 1990’s a Slocum glider prototype was ready, with wings and tail using vertical motion to glide horizontally and control its heading, moveable mass to control pitch and roll and thermal engine that enable great range and long missions.

The gliders of the 2000’s include the final development of the Slocum glider, whose battery is optimised for shallow water, where rapid turning and vertical velocity changes are needed. At this time Russ Davis’s Spray and the University of Washington Applied Physics’s Seaglider, for deep ocean, long duration missions with emphasis on energy efficiency were developed [2]. The Spray glider has a cylindric pressure hull with two wings and a vertical tail and unlike the Seaglider and the Slocum it uses two internal moving masses, one for roll located in the nose which can rotate 360 degrees and one for pitch which battery pack is moved by a rack and pin system driven by DC motors. It also has an external oil filled bladder for the ballast system and uses lithium batteries which has better energy density and performance than alkaline batteries. The Seaglider is designed for deep oceanographic sampling missions and has a range of 6000 m or 900 dives to 1000 m depth, it has an internal pressure hull designed to be isopycnal i.e. it has the same compressibility as the seawater which reduces the ballast pumping requirements and extend the vehicles range by as much as 50% compared to using a stiff hull [1]. The Seaglider’s external fairing is derived from a low draw laminar flow shape designed to reduce pressure drag with an advantageous pressure gradient at the rear end. The Slocum, Spray and Seaglider are reusable, inexpensive and operative in wide networks making them, and gliders as a whole, a superior tool in oceanography.

Reaching the present, Airbus Defence and Space has joined a sub-sea space robotics project called ROBEX (Robotic Exploration of Extreme Environments), here common space robotics technologies are utilised in an underwater glider application to foster the development by the synergy effects in these environments. The sting-ray shaped sub-sea glider, see Figure 2.1 [3], will like the previous gliders use buoyancy force to cycle vertically in saw-tooth patterns by changing the vehicle’s displaced volume, allowing low energy consumption, to enable the glider to operate for months in a single deployment whilst gathering various measurements from the deep ocean. However, unlike the conventional torpedo shaped Slocum, Spray and Seaglider, the ROBEX MOTH glider has a reduced attitude ability, it has a much higher area to mass ratio and the hydrodynamic effective area away from the centre of mass is much greater than for the previous mentioned gliders, especially in lateral direction.

Within the ROBEX alliance there has been previous versions of an underwater glider, although these were connected to the ground station with a cable, limiting the range of the mission to the length

---

2Partners within the ROBEX alliance are: Airbus DS, DLR, Marum, AWI, DFKI, GEOMAR, Helmholtz, Jacobs University, Unveristät Bremen, Technische Universität (TU) Berlin, TU Dresden, TU Kaiserslautern, TU Würzburg and TU München [3]
of the cable. Naturally, it is now of interest to progress, to reach greater depths and therefore to operate fully autonomously without a physical link to the ground station.

In the framework of this development, the objective with this master thesis is to support the Systems Engineering department at Airbus DS working on the ROBEX project, by designing and implementing the avionics system of the MOTH glider. This encompasses development of the Glider System’s Architecture, hardware and software, including interface definitions, state modelling, programming a sensor emulator for data communication as well as implementation and verification of a pitch controller.

Figure 2.1: ROBEX MOTH Deep-Sea Glider

3 Problem

The environmental conditions of the deep ocean prompts substantial challenges in the development of an autonomous underwater glider; taking cost, speed, limited energy supply and limited accommodation space inside the glider into account as well as developing an efficient communication system to operate with limited roaming meanwhile making the overall control system simple and flexible to enable a broad range of applications.

3.1 Objectives

The main objective of this thesis is to design and implement a glider control system and to work within a cross company collaboration and gain further knowledge in Systems Engineering strategies and deep-sea technologies.

Systems Engineering is a multidisciplinary field and a objective with this thesis is therefore also to gain an understanding of systems engineering tasks from different views. Using a top down approach, i.e. starting from a high level overview of the concepts of the system and the requirements followed by defining the system architecture, electrical architecture and OBC architecture, see the V-model presented Figure 3.1. At design level, operative system selection, state machine modelling and supporting
the CPU selection, at implementation level, pitch control modelling and creating a sensor emulator.

![Figure 3.1: The V-model](image)

### 3.2 Problem Description

The objective with the ROBEX alliance is to utilise synergy effects between space and deep sea research. In space, the available hardware that is radiation resistant is very limited whilst more powerful hardware exists for underwater applications. However, for underwater applications other design issues, e.g. isolation to protect from leakage, are increasingly important. In both environments there is very limited communication which pose further interest in advanced automatic systems, also when designing robotics for either application there is generally very limited space (volume) posing further difficulties. By implementing space technologies in underwater applications, the synergies could be utilised to enable design of reliable systems, established and documented systems and engineering processes that could optimise the hardware to be implemented in both deep-sea and space applications.

This thesis project includes an introduction to System Engineering procedures at Airbus DS, development of the control architecture, hardware and software, including interfaces, specification of control modules, electrical architecture as well as the implementation of the selected control modules. Also, the selection of a real time operative system, a multi-core processor and interfaces so that the overall system is as simple and flexible as possible, enabling measurement instruments to be swopped easily between missions of different characters.

Moreover, a pitch control study to model the glider’s sawtooth trajectory using wing flaps, a rudder, moveable mass and buoyancy tank. Last, a sensor emulator and OBC emulator for early simulations as a first step for the implementation of the developed system and as a part of the validation of the system design.

This thesis also includes test and verification of the implemented modules and documentation of the achieved results and outlook for the continuation of the activities.
3.3 Approach

Systems engineering is an interdisciplinary field of engineering that focus on how to design and manage convoluted engineering systems over their life cycle. To obtain knowledge from system top level to detailed component level, this thesis project will include several interdisciplinary tasks, see Figure 3.2.

The report is structured from top level down, starting with a systems system description including system architecture, behaviour and dynamics, followed by a subsystems and implementations. The OS and CPU selection, have been included in the electrical architecture chapter.

![Figure 3.2: System level breakdown with thesis tasks marked in yellow and blue](image)

4 System Description

The MOTH glider will be the first fully autonomous ROBEX glider, which will allow missions at greater depths and operations of longer duration than previous ROBEX gliders. The MOTH glider has a very high area to mass ratio compared to conventional torpedo shaped gliders and reduced attitude stability. The amount of hydrodynamic effective area from the centre of mass is, especially in lateral direction, is much greater than for e.g. Slocum, Spray and Seaglider, subsequently the lever arm of hydrodynamic wing forces is significantly larger than the restoring torque of gravity [4]. With this configuration the fin is closer to the centre of gravity than for the torpedo shaped gliders which makes it easier to control the yaw motion, although having no fin tail, add some difficulties for the pitch motion which in the torpedo set up could be stabilised using a passive fin tail. The lack of fin tail also consequent a significant roll-yaw coupling which can lead to an unstable attitude, all of which encourage the implementation of an active three axis attitude control of the glider. The primary objective of this control loop is thus to acquire and maintain a constant glider attitude throughout the entire trajectory during operation.

The glider receives GPS data on surface where it also transmits data via an Iridium link, it will also use telemetry and telecommunication (TM/TC) to communicate with the control station and is equipped with an independent emergency system using acoustic signals. After operation, data will be transferred through an Ethernet cable link at the ground station similar to previous models.
4.1 System Architecture

To meet the system requirements, identified in Appendix A, the glider will be equipped with a set of navigation sensors, actuators including moveable mass, buoyancy engine, a rudder, a left and a right wing-flap, remote unit terminals, cable interfaces, battery, memory, an independent emergency system, TM/TC unit and scientific measurement instruments/payload.

An architecture overview of the glider is illustrated in Figure 4.1, as shown, the ground control station and the on-board GPS and TM/TC unit receive and transmit data via satellite communication.

![Glider architecture overview](image)

**Figure 4.1: Glider architecture overview**

The on-board computer (OBC) shall control all active components of the glider, i.e. the navigation and status sensors, the actuators incl. buoyancy S/S, emergency emersion S/S, communication unit, PCDU and the scientific measurement instruments. An architecture overview of the OBC tasks is presented in Figure 4.2, incl. housekeeping, navigation, power control, mode management, attitude control, commands, failure, detection, isolation and recovery (FDIR), on-board time synchronisation (supported by GPS at surface), acquisition of telemetry and events, acquisition of sensor data, time stamped logging of events and sensor data, control of actuators, storage and retrieval of sensor and logging data in non-volatile memory and payload management and processing. A detailed device list for the OBC can be found in Appendix B.
4.2 Glider Behaviour

In software development and systems engineering a use case is typically a list or figure including steps to achieve a goal, this typically by defining a role, e.g. a human/user or external system, and a system. To represent the mission goals, a use case analysis was carried out at an early stage of the software development using the system modelling language (SysML). This study also had the objective to determine which subsystems and functionalities are needed in the glider to obtain the desired behaviour in different scenarios.

A high level use cases is presented in 4.3 of the Glider system, as seen from the figure, the glider needs to be prepared for the mission which includes setting up communication, system calibration, mission configuration and monitor which it also have to do during operation. During operation, when applicable, the control station will handle data, GNC calculations, actuators and if necessary the emergency system. Both before and after a mission has ended or been aborted maintenance is usually required including tasks such as downloading data, software modules and changing hardware and of course the control station is also responsible to recover the glider.

One level down, at subsystem level, the OBC use case is derived with actors: mission timeline handler, scheduler, database, release mechanism and actuators, see Figure 4.4. The mission timeline handler plans the trajectory, the scheduler reads and stores sensors and payload data in the database as well as controlling the GNC and emergency systems. This use case contains both include and extend links, the difference between these are that include connections are required whilst extend is optional,
e.g. triggering the emergency lift will hopefully not always be used.

Other use cases on different glider scenarios such as diving, initialising and recovery to name a few are available upon request.

4.3 Glider Dynamics

The glider is shaped like a Batoidea, more commonly known as rays, with a wing span of 2995.4 mm, height of 303.2 mm and length of 1878.7 mm, see Appendix C. The lateral control concept relies on two control surfaces: the left and right wing flaps and the rudder fin. The flap concept for the MOTH glider is implemented to operate as an aileron/elevon combination and the two wing flaps must therefore be
controlled independently. The aileron serves as an asymmetric part of the deflection of the wing flaps, e.g. if the left wing flap is at -2° and the right wing flap is at +4° then the aileron deflection would be +3° and elevon deflection of +1°.

The MOTH glider use buoyancy force, generated by filling and emptying an internal tank, to cycle vertically in saw-tooth patterns and operates with a maximum speed of 1 m/s, a maximum depth of 1000 m and for a maximum duration of 48 hours without recharging. As the glider will mainly be traveling in a straight line, trimming will only be required in the longitudinal direction and will be done using an internal moveable mass.

For the actuation parameters, the following parameters can be specified for the preliminary design:
- Actuation range: +/- 15° from neutral axis
- Positioning accuracy: command resolution 0.5° & maximum position error +/- 0.25°
- Positioning speed: 5°/s

The control surface dimensions fin size and characteristic length are yet to be specified. The dimensioning hydrodynamic parameters used to stabilise the glider are the restoring yawing torque due to a sideslip angle specified by the rudder coefficient and the rudder control torque which define the yawing torque due to a fin deflection angle. If the entire fin is used as a rudder, the rudder coefficient and the rudder control torque will be equal.

In Figure 4.5, a conceptual glider wing profile is presented which will be used as a coordinate system reference for the torque equations, the x-axis is simply aligned with the horizon, y-axis with the longitudinal axis perpendicular to the horizon and the z-axis with the vertical axis.

Assuming that the rudder/flaps are acting like a free surface, the required control torque, at a given flight regime and standard deflection, for a sufficient agility required to perform the manoeuvres, trajectory corrections, compensate external disturbances and accounting for the vehicle inertia, can be approximated as

\[ T_{\text{roll}} = I_{xx} \dot{\omega}_X \]  
\[ T_{\text{pitch}} = I_{yy} \dot{\omega}_Y \]  
\[ T_{\text{yaw}} = I_{zz} \dot{\omega}_Z \]

where \( I_{xx} \) is the moment of inertia around the x-axis and \( \dot{\omega}_X \) is the angular acceleration around the x-axis and

\[ T_{\text{pitch}} = I_{yy} \dot{\omega}_Y \]

where \( I_{yy} \) is the moment of inertia around the y-axis and \( \dot{\omega}_Y \) is the angular acceleration around the y-axis and

\[ T_{\text{yaw}} = I_{zz} \dot{\omega}_Z \]

where \( I_{zz} \) is the moment of inertia around the z-axis and \( \dot{\omega}_Z \) is the angular acceleration around the z-axis.

The equations of motions can be expressed as

\[ \dot{x} = v_x \cos(\theta) + v_z \sin(\theta) \]  
\[ \dot{z} = -v_x \sin(\theta) + v_z \cos(\theta) \]
\[ \dot{\theta} = \omega_y \]  

where \( v_x \) is the free stream velocity in the x-direction, \( v_z \) is the free stream velocity in the z-direction, \( \theta \) is the pitch angle and \( \omega_y \) is the angular velocity around the y-axis.

Additional to the \( x \) (horizontal), \( y \) (vertical) and \( z \) (longitudinal) coordinate system presented in Figure 4.5, a coordinate system, \( e_1, e_2, e_3 \), with the center of buoyancy and the chord line as reference has been added, see Figure 4.6. In this frame, \( e_1 \) is aligned with the chord line, \( e_2 \) vertically perpendicular to the chord line and \( e_3 \) longitudinal perpendicular to the chord line. The angle of attack, \( \alpha \), is the angle between the chord line and the free stream velocity, \( \theta \) is the pitch angle which is the angle between the chord line and the horizontal line and \( \varphi \) is the flight path angle i.e. the angle between the horizon and the free stream velocity. The drag force acts opposite to the free stream velocity and is perpendicular to the lift force.

![Figure 4.6: Coordinate system for longitudinal control calculations](image)

The kinetic energy \( E = \frac{1}{2} I \omega^2 \) can be utilised to reformulate the torque equations (1-3) as the torque is proportional to the product of the kinetic energy and the angle moved in radians i.e. in this case \( \beta + \delta_{\text{rud}} \) where \( \beta \) is the sideslip angle and \( \delta_{\text{rud}} \) the rudder deflection. The moment of inertia can be written as \( I = mr^2 \) where \( m \) is the mass of the actuator and \( r \) is the distance to its axis of rotation. The angular velocity can be expressed as \( \omega = \frac{v}{r} \), where \( v \) is the freestream velocity of the glider and \( r \) is the trajectory curve radius. The energy equation can thus be formulated as \( E = \frac{1}{2} \rho v^2 x A \) where \( \rho \) is the water density, \( v \) is the glider velocity \( x \) is the lever arm, i.e. the distance between the glider centre of mass and the hinge point/ center of rotation and \( A \) is the surface area of the actuator. The roll, pitch and yaw torques generated by the actuators, under the assumption that the main effect of the trailing edge flap is the extra lift created by the flap area due to deflection of the flap with regards to angle of attack, can be formulated as

\[ T_{\text{roll, flap}} = \frac{\rho}{2} v^2 C_{L\alpha}(\alpha + \delta_{\text{flap}}) y_{\text{flap}} A_{\text{flap}} \]  

(7)

where \( C_{L\alpha} \) is the lift coefficient with regards to the angle of attack, \( \alpha \) is the angle of attack of the glider, \( \delta_{\text{flap}} \) is the flap deflection and \( y_{\text{flap}} \) is the width of the flap and \( A_{\text{flap}} \) the flap surface area. The flap pitch torque can be computed as

\[ T_{\text{pitch, flap}} = \frac{\rho}{2} v^2 C_{L\alpha}(\alpha + \delta_{\text{flap}}) x_{\text{flap}} A_{\text{flap}} \]  

(8)
where $x_{\text{flap}}$ is the length along the chord line from the hinge point to the center of buoyancy (CB) i.e. where the resultant of all buoyant forces are assumed to act which can also be seen as the center of mass of the displaced water.

$$T_{\text{yaw, rud}} = \frac{1}{2} \rho v^2 C_{L\alpha}(\beta + \delta_{\text{rud}}) x_{\text{rud}} A_{\text{rud}}$$

(9)

where $x_{\text{rud}}$ is the length along the chord line from the trailing edge to CB, $\delta_{\text{rud}}$, and the sideslip angle, $\beta$ [4].

For acceleration, Newton’s second law of motion gives

$$\ddot{x} = \frac{F_x}{m}$$

(10)

$$\ddot{z} = \frac{F_z}{m}$$

(11)

where $m$ is the total mass of the glider and $F_x$ and $F_z$ are the horizontal and longitudinal forces calculated in equation (13) and (14). The second derivative of the flight path angle gives

$$\ddot{\theta} = \dot{\omega}_y$$

(12)

where the angular acceleration $\dot{\omega}_y$ is presented in equation (2) as equivalent to the fraction of the total torque around the y-axis and the moment of inertia around the y-axis.

The hydrodynamic forces and moment are calculated as

$$F_D = \frac{1}{2} \rho v^2 C_D(\alpha) A$$

(13)

$$F_L = \frac{1}{2} \rho v^2 C_L(\alpha) A$$

(14)

$$M_{DL} = \frac{1}{2} \rho v^2 C_m(\alpha) A$$

(15)

where $C_D$, $C_L$ and $C_m$ are the aerodynamic drag, lift and moment coefficients and $A$ is the maximum cross section area of the glider. Summarising the forces acting on the glider in horizontal (x-axis) and longitudinal (z-axis) direction gives

$$F_x = F_L \sin(\theta) - F_D \cos(\theta)$$

(16)

$$F_z = B - F_L \cos(\theta) - F_D \sin(\theta)$$

(17)

The glider is assumed to be designed such that the buoyancy force acting on the glider as a whole is of the same magnitude as the gravity force acting on the glider, these vertical force contributions are thus neglected from the model and only the additional buoyancy force generated by the tank remains in the longitudinal force equation.

The attitude control is implemented by the means of three hydrodynamic control surfaces that encompass a pair of trailing edge wing flaps placed at the outer area of the wing span and a steerable
Design and Implementation of a Glider Control System

tail fin acting as a rudder. In addition, an internal moving mass actuator providing longitudinal trim in order to reduce the power demanded by the elevon surface actuators is required also to balance the longitudinal torques acting on the glider. The actuator concept is summarised in Table 4.1.

Table 4.1: Glider Actuator Concept

<table>
<thead>
<tr>
<th>Euler Angle</th>
<th>Flap Concept</th>
<th>Implemented by</th>
<th>Trim Actuator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roll (x-axis)</td>
<td>elevon</td>
<td>left &amp; right wing flap</td>
<td>not required</td>
</tr>
<tr>
<td>Pitch (y-axis)</td>
<td>aileron</td>
<td>left &amp; right wing flap</td>
<td>longitudinally moving mass</td>
</tr>
<tr>
<td>Yaw (z-axis)</td>
<td>rudder</td>
<td>steerable tail fin</td>
<td>not required</td>
</tr>
</tbody>
</table>

Implementing this to control theory, the glider will of course use a closed loop transfer function where the Euler angles (roll, pitch and yaw) are the measured output $y(t)$ of the actuator system and $r(t)$ is the reference i.e. the desired output based on a predefined trajectory. The controller $C$ then takes the error, see equation (20), to change its inputs $u(t)$ to the system $P$, see Figure 4.7.

![Figure 4.7: General closed-loop controller](image)

The controller $C$, the system $P$ and the navigation sensors $F$ are linear and time-invariant, subsequently, the control loop can be analysed using the Laplace transforms:

$$Y(s) = P(s)U(s)$$  \hspace{1cm} (18)

$$U(s) = C(s)E(s)$$  \hspace{1cm} (19)

$$E(s) = R(s) - F(s)Y(s)$$  \hspace{1cm} (20)

In Chapter 7, a model based feedback controller for the MOTH’s pitch control is presented encompassing both the dynamics and control theory presented in this section.

5 Electrical System Architecture

Once the components needed to meet the functional requirements have been identified the next step is to look at the interfaces between these. The glider has a set of actuators: a rudder, a left and a right wing flap, a moveable mass and a buoyancy engine, which are connected to the OBC through a CAN bus system, to minimise the overall complexity. The CAN bus interface has also been selected as the interface for the navigation sensors as well as to the temperature sensors and the communication
unit containing an RF Transponder, Iridium unit and acoustic transceiver. The CAN bus is also used to interface the remote terminal units (RTUs), there will be one RTU per tube and three tubes in total. The tubes carry the OBC, the AHRS, the movable mass, batteries, cables, connectors etc. and the RTUs are the border between the avionic’s system and the scientific instruments. The scientific instruments must be exchangeable, therefore configurable interfaces are required (established by the RTUs) whereas the system sensor suite is more static. As seen in the electrical system architecture presented in Figure 5.1, several components are connected to an RS232-CAN converter to be compatible with this setup. All communication, actuation, navigation components are directly connected to the power supply whilst the scientific instruments are to be supplied with power via the RTUs through RS232 interfaces. The independent emergency system will be connected to the OBC through an I/O interface and be run by autonomous power with external/onboard activation and connected to an acoustic sensor. The GNS receiver and flight recorder will be integrated to the OBC directly. The RF transponder contains an 868 MHz antenna and the Iridium and GNS receiver uses an SUB GPS/I-1300M antenna. When possible the OBC will be connected to the ground control station via an Ethernet cable and a LED flasher will be installed to ease the recovery.
5.1 CAN bus, RS232 & Ethernet Cable link

The bus system is still under debate as of April 2015, the electrical architecture overview presented in Figure 5.1 is the most recent set up. The aim is to make the system as simple as possible by minimising the amount of unique components, therefore trying to connect all components to a CAN bus interface, with RS232-CAN converters to the components that required a different (RS232) interface and an Ethernet link for usage at the ground station.

The RS232 is a standard for serial communication and defines the signals connecting between a data terminal equipment and a data communication equipment. The RS232 serial port however has the disadvantage of low transmission speed, large voltage swing, and large standard connectors.

The Ethernet cable link allows the ground station computer to communicate with the ASW when RF and acoustic means are not of preference. Ethernet is a common local area network (LAN) technology that support high bit rates and longer link distances. Systems communicating via ethernet divides data into frames that contains source and destination addresses and checks for errors in the data which is re-transmitted when an error is detected. Ethernet cables are however limited in distance due to their electrical transmission characteristics but are easy to use, secure, reliable which minimise communication issues and are good for stabilisation and tend to offer better performance and protection against electrical interference than other networking technologies.

5.2 Sensor & Actuator Data

All navigation sensors shall collect data autonomously without ASW intervention and the selected OBC driver shall communicate with the hardware according to the interface standards. The BSW shall store all the incoming data internally and shall make it available through a service call. The sampling frequency is fixed prior run-time and the data not received by the ASW before a new set is acquired will be lost.

As seen in Figure 5.1, the Navigation Sensors includes an Attitude & Heading Reference System (AHRS), which provides status data, time tag and a set of 3-axis attitude angles, a Global Navigation Sensor (GNS), which provides the glider position with regards to the global navigation coordinate system and shall acquire a status signal, as a minimum the status of the connectivity to the navigation satellites, and last a Pressure sensor, providing depth information with time tag.

The actuator data is collected by the ASW and passed to the actuator interface in SI units using a service call. The ASW shall be able to send position commands to the left wing flap actuator, the right wing flap actuator and the tail fin/rudder actuator, all with regards to the zero degree position. The ASW shall also send commands to the Emergency Mass Release (EMR) which interface is envisaged to be a microprocessor interfacing with the OBC.
5.3 Science Data & RTUs

The acquired science data is collected via the Remote Terminal Units, which provide the instruments with power and data services, to the OBC memory. The ASW will power up the instruments, send initialisation and configuration data, download science data from RTU and start/stop data acquisition. This RTU interface setup enables a more flexible system as the instruments easily can be swapped, however, this will require more power in order to also operate the RTUs.

5.4 Telemetry & Telecommunication

No telemetric transmissions are sent or received when submerged. However, whilst on surface, the OBC will send a predefined set of housekeeping data from the run-time memory to the acoustic and RF transceiver interfaces when the corresponding service is called by the ASW.

Telecommands can be sent to the OBC via the RF interface when the glider is at surface or via acoustic interface when submerged. The envisaged telecommands regard health status data to ground station via RF link, updates of GNC parameters and command to trigger the EMR. The OBC shall scan the telecommand interfaces for new incoming telecommands without intervention of the ASW and store this data in a FIFO buffer in the BSW.

5.5 On-Board Computer

The OBC is used to host all data, connect with the sensors, measurement instruments, etc. and needs to be powerful enough to compute all calculations. To select the most favourable board for the MOTH glider criteria such as price, RTOS capability, CPU performance (high enough to finish the MOTH computations in time), multicore processor, on-board flash (req. enough storage), on-board RAM, power, sleep mode (to save power), RTC (to keep track of current time), support Ethernet, CAN and RS485 interface, see Appendix D for OBC trade-off. Based on weighing these criteria the GX-6300 computer was selected, having the highest cumulated points after the trade off. This industrial computer is an ARM based embedded Linux computer for industrial applications. It has a sufficient 400 MHz ARM platform, 256MB Flash, 128MB RAM, 4MB NOR Flash, Ethernet interface, a set up of three CAN bus interfaces and two RS232 interfaces to meet the communication requirements according to the electrical architecture, further GX-6300 details are available by contacting Axotec³ [5].

5.6 Operating System

To manage the OBC hardware and software and provide common services an operative system is required, in particular, to guarantee it operates according to a predefined time schedule a real time

³Contact: Axotec Technologies GmbH, info@axotec.de, +49 (0)8171 92192-0
operative system is required. Moreover, a preemptive multitasking system is requested i.e. a system that operates and executes tasks with regard to priority and can temporarily interrupt a task and resume first after tasks of higher priority have been carried out, see Figure 5.2 for an applicable MOTH example. As seen, a log system, for example, can be interrupted without requiring its cooperation if the GNC system is triggered as it has a higher priority.

![Figure 5.2: Example of preemptive scheduling](image)

Included in the operative system trade off were LynxOS, QNX Neutrino, RTLinux, Linux Debian, VxWorks, PikeOS, FreeRTOS, RTEMS and Integrity. The criteria in the selection were:

- Supported CPU Architectures (e.g. x86, PowerPC, ARM, etc.)
- OpenSource / Commercial?
- Costs for a development or runtime license
- Easy Support / Knowledge Database Size?
- US-based company?
- Support of Time & Space Partitioning (ARINC 653)
- Any Development Aids (e.g. Development Kits, Tutorials, IDE)

Based on weighing these criteria, the VxWorks, Pike OS and Linux Debian obtained the highest scores, see Appendix E for trade off chart, the first two mentioned are commercial operative systems which in an industrial use could have been more appropriate. When a Linux CPU board had been selected it made the OS selection simple: the open source Linux Debian for its CPU compatibility and free of charge, suit this research application.

### 5.7 Solid State Memory

During execution of the ASW, the OBC shall log a predefined set of data to the SSM without intervention from the ASW. It shall only be possible to download and clear the SSM through the Ethernet cable link interface when connected to ground station.
6 System State Modelling

A state machine describes how a system behaves and could help to identify possible control design errors, e.g. if there are any possible states where the glider is not able to exit after entering or if any necessary transitions are missing for reaching desired nominal behaviour, and was designed for the MOTH glider. In general, a state machine stores the status of something at a given time and can operate based on inputs to change this status and/or cause an action or output to be executed or sent out for any given change. A state machine therefore contains an initial state, a set of possible input and output events, a set of new states that may be entered based on the inputs and a set of possible actions or output events issued from the new state.

6.1 Design

The states and events included in the MOTH System state machine are presented in Table 6.1 and Table 6.2.

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>The system is completely off, consumes no power and returns to working state only after rebooting.</td>
</tr>
<tr>
<td>Initialisation</td>
<td>System locates and use the initialisation files (files with INI suffix in Windows) to find definite values to substitute for variable values.</td>
</tr>
<tr>
<td>Checkout</td>
<td>Actuators and sensors are turned on and payload, communication and system are checked.</td>
</tr>
<tr>
<td>Preparation</td>
<td>Parameters are set, trajectory is loaded, trim</td>
</tr>
<tr>
<td>StandBy</td>
<td>System is fully usable, awaiting command to start mission</td>
</tr>
<tr>
<td>Operational</td>
<td></td>
</tr>
<tr>
<td>- Surface</td>
<td>Glider is at surface, get GPS position, send and receive data, control active, follow timeline</td>
</tr>
<tr>
<td>- Descend</td>
<td>Change pitch and glide angle, correction with actuators</td>
</tr>
<tr>
<td>- Transition</td>
<td>Buoyancy change, control pitch attitude</td>
</tr>
<tr>
<td>- Ascend</td>
<td>Change pitch and glide angle, correction with actuators</td>
</tr>
<tr>
<td>NonOpsError</td>
<td>At surface, start-up (non operational) error encountered, proceed by following the defined FDIR strategy</td>
</tr>
<tr>
<td>Maintenance</td>
<td>Repair and check system with external support at ground station</td>
</tr>
<tr>
<td>Salvage</td>
<td>At surface, recovery of glider: read and transmit GPS position to ground station, await ship to collect glider</td>
</tr>
<tr>
<td>OpsError</td>
<td>In water, operational error encountered, hold-up operation, follow defined FDIR strategy</td>
</tr>
<tr>
<td>Emergency</td>
<td>In water, lethal error encountered, release weight, abort mission</td>
</tr>
</tbody>
</table>
Table 6.2: List of Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power On</td>
<td>Power switched on</td>
</tr>
<tr>
<td>Power Off</td>
<td>Power switched off</td>
</tr>
<tr>
<td>Reset</td>
<td>clear to restore system to zero, i.e. set system again</td>
</tr>
<tr>
<td>Start Checkout</td>
<td>System enabled, go to Checkout state</td>
</tr>
<tr>
<td>Error</td>
<td>Error encountered, hold-up state events until system is restored to nominal</td>
</tr>
<tr>
<td>Start Maintenance</td>
<td>Go to Maintenance</td>
</tr>
<tr>
<td>Start Preparation</td>
<td>System initialised, go to Preparation state</td>
</tr>
<tr>
<td>Arm System</td>
<td>System ready, go to standby state</td>
</tr>
<tr>
<td>Disarm System</td>
<td>Hold-up from starting mission, go to preparation</td>
</tr>
<tr>
<td>Start Mission</td>
<td>Go to Operational Surface state</td>
</tr>
<tr>
<td>Dive</td>
<td>Go to Descend state</td>
</tr>
<tr>
<td>Rise</td>
<td>Go to Ascend state</td>
</tr>
<tr>
<td>Surface Reached</td>
<td>Go to Surface state</td>
</tr>
<tr>
<td>Start Transition</td>
<td>Go to Transition state</td>
</tr>
<tr>
<td>End of Mission</td>
<td>Reached end of trajectory, go to Salvage state</td>
</tr>
<tr>
<td>Recover</td>
<td>System restored to nominal state, return to Operational state</td>
</tr>
<tr>
<td>Not Recoverable</td>
<td>System unable to restore to nominal state, go to Emergency state</td>
</tr>
<tr>
<td>Abort Mission</td>
<td>Critical failure encountered, go to Emergency state</td>
</tr>
<tr>
<td>Retrieved</td>
<td>Glider has been collected, enter NonOpsError if retrieved triggered by mecha-</td>
</tr>
<tr>
<td></td>
<td>nical button after being collected, enter Checkout if retrieved from sent GPS</td>
</tr>
<tr>
<td></td>
<td>location and or LED flash</td>
</tr>
</tbody>
</table>

The initial state is the off state which is demonstrated in the state machine, see Figure 6.1, and to leave this state and enable the system, the power needs to be switched on. This change the state to the initialisation process that is to power up/ boot the system, read initialisation files and define values. In case of failure, the system reboots. After completing the initialisation successfully, the OBC start to run the checkout commands where the actuators and sensors are switched on and the control, payload and communication is checked. Incase of encountering an error, the system is diverted to the non-operational error state where the predefined FDIR strategy is executed to restore the system to nominal state or directly to maintenance. If unable to recover in the non operational state, the system will also be sent to the maintenance state where the system is either repaired or taken out of service. If repaired here, the system will re-enter the checkout state and re-run the internal actions of this state.

In the preparation state, the parameters are set and the trajectory is loaded, again incase of an error encounter the system will be directed to either non operational error or directly to maintenance, according to the same procedures as mentioned for the checkout error. When the preparation actions are completed, the system will be ready to enter standby state to await the start of the mission.
In the operational state, an internal initial state has been defined connected to the surface state, where the glider collects the GPS position and sends and receives data. The surface state could be reached several times during operation, however at the end of the trajectory this will trigger an end mission command and the salvage state can be entered. In all other surface state entries, i.e. when still not at the end of the trajectory, a transition state is reached leading to a buoyancy change to start descending. In the descend and ascend states, which are entered after a buoyancy change, the actuators are controlled to navigate according the loaded trajectory. The descend state reach an exit point when the target depth is reached and the glider will therefore transition and then start to ascend. It is not necessary to ascend all the way to the surface at each ascend entry, the ascend and the descend limits will depend on the trajectory which will also defines the requested number of the re-entries of the surface state before the glider has reached the trajectory’s end point.

At the end of the trajectory, at surface, the glider enters the salvage state where the GPS signal is transmitted and the system awaits pick-up and then goes through a checkout. However, in the case of operational failure during the mission the system will put the operation on-hold and enter the operational error state where the FDIR strategy is executed. This could trigger the emergency emersion if unsuccessful i.e. non-recoverable, however, if restored to nominal state, the mission would continue.

If the emergency system is triggered, a weight is released to directly bring the glider to surface, here it enters salvage state if it is able to detect the surface, send its GPS coordinates or activate its LED light. If unable to signal its location it would however require to be collected by a possibly more extensive search mission and when retrieved this status would be triggered manually and the glider would be brought to non-operational error/maintenance.

6.2 State Control Signalling

To run the finite real time state machine presented in Figure 6.1, a function that maps states and inputs to outputs is of course required as well as a function that maps states and inputs to states. This will be implemented in a mode controller that observes the current state and inputs, e.g. if the pressure sensors informs that the surface is reached this would change the current for example ascend state to the transition phase, or if reaching the surface in the emergency state after releasing a weight, detecting the surface would change the state to salvage. The mode controller would run an infinite loop with some of the glider systems running periodically, e.g. motion/path planning, whilst some systems will wait for a command/get called before running , e.g. the actuators. The mode controller checks if a mode change (state change or action) is allowed and distributes this information to all components. Thereafter, each component, as they behave differently in different states, decides if it is justifiable to change behaviour. For example, when in descend state it would not make sense to communicate via satellite, which should be disabled in this state, therefore it should be prohibited in the control
functions for the glider to try to communicate via satellite in this state.

Figure 6.1: MOTH System State Machine
7 Pitch Control

Pitch control shall be provided by a horizontally aligned control surface (perpendicular to the rudder), the moveable mass, the buoyancy tank and the fixed internal weight, all of which creates pitch torque. This torque shall be used to control the pitch angle of the vehicle. The nominal pitch angle depends on the state (ascend, descend, transition) and the angle of attack variations create variable pitching moments which need to be compensated. Filling or emptying the buoyancy tank shifts the centre of buoyancy which creates a pitching torque that is modified by variations of the vehicle velocity. External perturbations from the surrounding environment (changes in horizontal or vertical water currents) also have to be compensated. Pitch trimming shall be provided by the moving mass which shall be operated such that the actuator torque of the elevon is zero during the stationary glide phases.

7.1 Equations for Longitudinal Control

A two dimensional model has been created which contains equations of motion for two translational and one rotational degree of freedom \((x, z, \theta)\) presented in section 4.3, the model also contains the buoyancy torque as function of displaced tank volume (for simplification the tank is either completely full or empty), the moving mass torque and fixed mass torque. The different masses and their positions are conceptually presented in Figure 7.1. The stationary mass \(m_s\) is the sum of the distributed hull mass, \(m_h\), the fixed mass, \(m_w\), and the ballast (buoyancy tank) mass, \(m_b\). The total mass \(m\) is thus \(m = m_s + \bar{m}\) where \(\bar{m}\) is the internal moveable mass. Each mass is defined with a length vector \(r\) to the center of buoyancy (CB), with corresponding indices. The position of the center of buoyancy varies with the pitch angle so the distances to this point are thus not constants.

The buoyancy force contribution from the tank can be calculated as

\[
B = \rho g \Delta V
\]  

(21)

where \(\Delta V\) is the change in volume i.e. the displaced volume in the buoyancy tank. This force could also be expressed as the product of the ballast mass, \(m_b\), and the gravity acceleration. The tank’s torque contribution can then be computed with regards to the distance to the center of buoyancy, \(r_b\), as

\[
T_{pitch,B} = r_b \times B
\]

(22)
The torque contribution from the movable mass can be written as

\[ T_{\text{pitch,}\bar{m}} = \vec{r} \times \vec{m}g \] (23)

where \( \vec{r} \) is the distance from the movable mass, \( \bar{m} \), to the center of buoyancy.

The torque contribution from the fixed mass can be written as

\[ T_{\text{pitch,}m_w} = r_w \times m_wg \] (24)

where \( r_w \) is the distance from the fixed mass, \( m_w \), to the center of buoyancy as shown in Figure 7.2.

The glider’s pitching moment about the center of buoyancy can be computed as

\[ M_{\text{pitch}} = M_{DL} + M_{DL,eq} \] (25)

where \( M_{DL,eq} \) is the moment for which the system is in equilibrium and it can be computed around the center of buoyancy as

\[
M_{DL,eq} = (m_{e3} - m_{e1})v_{e1}v_{e3} - \bar{m}g(\bar{r}_{e1} \cos(\theta) + (\bar{r}_{e3} \sin(\theta)) - \\
m_wg(r_{b,e1} \cos(\theta) + r_{b,e3} \sin(\theta)) - m_wg(r_{w,e1} \cos(\theta) + r_{w,e3} \sin(\theta))
\] (26)

where \( m_{e1} \) and \( m_{e3} \) are the added mass terms derived by Kirchhoff [6]. The added masses are included because of the inertia added to the system as an accelerating body in fluid must deflect some volume of the surrounding fluid as it moves through it. The distributed hull mass is not included in equation (26) as it has no lever arm with regards to the centre of buoyancy.

The total pitch torque acting on the glider, presented in equation (2), is the sum of the pitch torque contributions from all actuators, i.e the flaps (8), buoyancy tank (22), moveable mass (23) and fixed mass (24), and the glider as a whole (25).

### 7.2 Model-Based Feedback Control

A pitch controller with three degrees of freedom (\( x, z \) and \( \theta \)) was created in Matlab using the equations described in section 4.3 and 7.1. It is a reduced model based on the longitudinal plane and considering a uniformed distribution of mass as well as defining the buoyancy tank to always be either completely full or completely empty. The torque contribution of the moveable is calculated based on it’s distance to the center of buoyancy which for simplification has been decided to to only vary based on the pitch, in reality the mass will also move in \( e_1 \)-direction up to 20 cm [TBD].

The MOTH glider’s parameters are yet to be decided, in order to run a simulation of the longitudinal control parameters were taken from the Slocum glider, see Table 7.1 [6]. The *-marked values are not taken from Slocum and the controller gains, \( K_P \) and \( K_D \), were tuned to try to obtain a controller...
Table 7.1: Parameters

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m [kg]</td>
<td>52</td>
<td>total mass</td>
</tr>
<tr>
<td>mw [kg]</td>
<td>1</td>
<td>fixed mass</td>
</tr>
<tr>
<td>mh [kg]</td>
<td>40</td>
<td>hull mass</td>
</tr>
<tr>
<td>mb [kg]</td>
<td>0.5</td>
<td>full tank mass</td>
</tr>
<tr>
<td>mp [kg]</td>
<td>9</td>
<td>moveable mass</td>
</tr>
<tr>
<td>me1 [kg]</td>
<td>5</td>
<td>added mass</td>
</tr>
<tr>
<td>me3 [kg]</td>
<td>70</td>
<td>added mass</td>
</tr>
<tr>
<td>Iyy [kg/m²]</td>
<td>12</td>
<td>moment of inertia around y-axis</td>
</tr>
<tr>
<td>A [m²]</td>
<td>0.036</td>
<td>glider cross section area</td>
</tr>
<tr>
<td>Af [m²]</td>
<td>0.156*</td>
<td>surface area flaps</td>
</tr>
<tr>
<td>xc [m]</td>
<td>1.5*</td>
<td>characteristic length of glider</td>
</tr>
<tr>
<td>Kd0</td>
<td>2.15</td>
<td>initial drag constant</td>
</tr>
<tr>
<td>Kd</td>
<td>24.95</td>
<td>drag constant</td>
</tr>
<tr>
<td>Kl</td>
<td>132.55</td>
<td>lift constant</td>
</tr>
<tr>
<td>Km</td>
<td>-100</td>
<td>moment constant</td>
</tr>
<tr>
<td>kp</td>
<td>0.001*</td>
<td>proportional gain</td>
</tr>
<tr>
<td>kd</td>
<td>0.1*</td>
<td>derivative gain</td>
</tr>
</tbody>
</table>

output that gave the desired trajectory results, the gravity constant was set to 9.81 m/s² and the density of the water to 1000 kg/m³.

The implemented control laws are based on the current depth. When \( z = 0 \), which is put at surface with positive \( z \)-axis pointing down in the water, the glider should start diving and the conditions for this is a full tank and a positive pitch (i.e. pointing down). The desired pitch is set to 30° whilst the actual pitch is calculated from the dynamics equations, as the glider is intended to travel in sawtooth patterns the pitch gradation in the transition points are neglected so that the desired pitch is always a constant (giving the sawtooth pattern which in reality would be more circular in the transitions) and therefore the desired pitch derivative is always zero.

The desired sawtooth trajectory needs to be bounded between an upper and lower altitude, these can be set to any values within the glider’s capability and depending on the mission. In the simulations, the upper boundary was set to \( z = 10 \), i.e. 10 meters below surface, and the lower boundary is put at \( z = 50 \), see Figure 7.2. When the glider reach either of these altitudes, the buoyancy tank is triggered to get filled (\( z=10 \)) or emptied (\( z=50 \)), and the glider then enters the transition, dotted area in Figure 7.2. By filling the tank at the upper boundary, the glider should start descending and the buoyancy force should, based on the defined coordinate system with positive \( z \)-axis pointing down, be positive (pointing down). At the lower boundary, the tank is emptied changing the buoyancy force to be
negative, i.e. pointing up towards the surface, and the glider starts ascending. To control this with respect to the sign, the buoyancy tank volume is set to

$$\Delta V = \pm \frac{m_b}{2\rho}$$

depending on if its ascending (−, with regards to z) or descending (+, with regards to z), which gives a change of volume between these states equal to the full volume.

To correct the deviance of the measured pitch angle from the desired pitch angle with the flaps, the deflection angle of the flaps is computed using a PD controller as

$$u = K_P(\theta - \theta_r) + K_D(\dot{\theta} - \dot{\theta}_r)$$

where $u$ is the input to the flaps, $K_P$ and $K_D$ are running parameters for controlling the proportional and derivative gain respectively, $\theta$ is the pitch angle, $\dot{\theta}$ the pitch derivative or the angular velocity around the y-axis, $\theta_r$ and $\dot{\theta}_r$ are the desired values for the sawtooth trajectory.

### 7.3 Simulink block diagram

The controller was designed in Simulink, see Figure 7.3, and consists of four Matlab function blocks, each connected to their own script with defined inputs and outputs. Comparing with the general closed-loop controller presented in section 4.3 the desired values are the references which compared with the values reported back from the feedback gives an error that is compensated for in the controller which sends an input to the dynamics where the equations of motion: velocities ($\dot{x}, \dot{z}, \dot{\theta}$) and accelerations ($\ddot{x}, \ddot{z}, \ddot{\theta}$) are defined. Between the control and dynamics blocks there is a limit block which puts boundaries on the variables e.g. $z$ is limited to positive numbers as the vehicle is not intended to be flying above the water, and the flaps are limited to a maximum deflection of $\pm 15^\circ$ according to the system requirements so that the input values to the system dynamics block are possible. The outputs from the dynamics then go into the integrator which find the solution to our equation of motions and generates back the integrated position and velocity values to the controller and so the loop continues.

There is also a feedback from the limits output back to the controller where only the tank volume and deflection angle are collected as these are not to go through the integrator and the controller needs updated input values of these to run the simulation. Last, a timer is connected to the controller to have control of the time and all variables are collected into plot blocks. The save function block saves
the computed variables and transforms the angles from radians to degrees before sending them to the plot matrix q.

![Matlab Simulink block-diagram of the designed closed loop feedback controller](image)

Figure 7.3: Matlab Simulink block-diagram of the designed closed loop feedback controller

To describe the flow of variables within the block diagram further see Table 7.2.

Table 7.2: Simulink block inputs and outputs

<table>
<thead>
<tr>
<th>Function</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>$x, z, \theta, \dot{x}, \dot{z}, \dot{\theta}, \delta, \Delta V, \theta_r$</td>
<td>$x, z, \theta, \dot{x}, \dot{z}, \dot{\theta}, \delta, \Delta V, \theta_r, e_\theta, e_\dot{\theta}$</td>
</tr>
<tr>
<td>Limits</td>
<td>$x, z, \theta, \dot{x}, \dot{z}, \dot{\theta}, \delta, \Delta V, \theta_r$</td>
<td>$x, z, \theta, \dot{x}, \dot{z}, \dot{\theta}, \delta, \Delta V, \theta_r$</td>
</tr>
<tr>
<td>Dynamics</td>
<td>$\dot{x}, \dot{z}, \dot{\theta}, \delta, \Delta V$</td>
<td>$\ddot{x}, \ddot{z}, \ddot{\theta}, \delta, \Delta V, T_p, T_b, T_w, T_f$</td>
</tr>
<tr>
<td>Integrator</td>
<td>$\dot{x}, \dot{z}, \dot{\theta}, \ddot{\theta}$</td>
<td>$x, z, \theta, \dot{x}, \dot{z}, \dot{\theta}$</td>
</tr>
<tr>
<td>Save</td>
<td>$x, z, \theta, \dot{x}, \dot{z}, \dot{\theta}, \delta, \Delta V, T_p, T_b, T_w, T_f$</td>
<td>$x, z, \theta, \dot{x}, \dot{z}, \dot{\theta}, \delta, \Delta V, T_p, T_b, T_w, T_f, \varphi, \alpha$</td>
</tr>
<tr>
<td>$e$</td>
<td>$\theta - \theta_r, \dot{\theta} - \dot{\theta}_r$</td>
<td>$x, \theta, \dot{x}, \dot{z}, \dot{\theta}, \delta, \Delta V, T_p, T_b, T_w, T_f, \varphi, \alpha$</td>
</tr>
<tr>
<td>$q$</td>
<td>$x, z, \theta, \dot{x}, \dot{z}, \dot{\theta}, \delta, \Delta V, T_p, T_b, T_w, T_f, \varphi, \alpha$</td>
<td></td>
</tr>
</tbody>
</table>

The developed simulator is available upon request\(^4\) and contains seven files with different extensions:

- `parameter.m`, must be run before starting the simulation and defines all constant parameters
- `simulation.slx`, Simulink block diagram that simulates the response of the glider under the chosen control laws, dynamics and limits
- `control.m`, defines the control laws for controlling the pitch angle, flap angle and buoyancy tank based on the depth.
- `limits.m`, defines the system boundaries
- `dynamics.m`, defines the equations of motions described in section 7.1

---

\(^4\)Contact: Hannah Lindberg, hannahli@kth.se, +4917683324248
Design and Implementation of a Glider Control System

- `save.m`, saves the variables for plotting and transforms angles from radians to degrees
- `plot.m`, needs to be run in Matlab after running the simulation to plots the results

7.4 Results

The results from the pitch control simulation are presented in this section, as the MOTH parameters are yet to be announced this model consists of several estimated values: for lift and drag constants, position of the masses, flap dimensions and of course our proportional and derivative constants which are varied to find the best model given it’s uncertainty. For pedagogical reasons the z variable is plotted negatively (axis pointing up towards the surface) showing that the glider is diving although in the model these values are positive (z-axis pointing down in the water). In the shown results the proportional gain, $K_P$, was set to 0.001, and the derivative gain, $K_D$, to 0.1.

In Figure 7.4, the glider displacement in x- and z-direction is plotted against time and shows that the glider is travelling in positive x-direction and descending as desired between the interval $z=0$ to $z=50$ meters. In the lower subplot of the displacement in z-direction it is also shown that the flaps are trying to compensate the pitch angle when it deviates from the desired value as its not keeping a constant pitch angle/linear trajectory.

The displacement in the x-z plane, see Figure 7.5, shows that the glider is diving until it reach the lower boundary at 50. At this altitude, which is more evident when running the simulation for a longer duration, the system becomes unstable while it tries to transition to an ascend state.

The pitch angle, see Figure 7.6, is within bounds until the glider starts to transition and then

![Figure 7.4: Glider displacement in x- resp. z-direction](image)
The pitch error, see Figure 7.7, is the difference between the calculated pitch angle and the desired pitch angle. As the desired pitch angle is $\pm 30^\circ$ and the calculated pitch angle is unstable the error merely shows the same behaviour with a $-30^\circ$ difference.
The free stream velocity is shown to increase as the glider is descending and slowing down when the flaps are trying to compensate a too high pitch, the flight path angle is the difference of the pitch angle and the angle of attack, i.e. two unbound values, see Figure 7.8.

The torque contributions of the different actuators all stabilise at different values and in the tank plot it is evident that once the tank was emptied at $z = 50$ there is a sign shift of the ballast torque, see Figure 7.9.
Design and Implementation of a Glider Control System

Figure 7.9: Actuator torques

The angular velocity, see Figure 7.10, show a change sign to increase and decrease in order to stay on the desired path however grows out of bound as well when the system reach a transition point.

Figure 7.10: Angular velocity
8 Software Architecture

The glider will be equipped with navigation sensors and scientific measurement instruments requiring corresponding application and basic software to be implemented in the OBC software that is in charge of controlling and monitoring the glider system. The OBC software architecture will consist of an application software (ASW) layer performing different tasks written in a high level programming language and a software interface to the basic software (BSW) service interface layer. The BSW provides a unique interface which allows to use the identical application software in a non real time simulation and the actual flight hardware which eliminates the software risk usually associated with software transfers between platforms. In the BSW layer, the operative system, scheduler and drivers are included as well as a service layer to the ASW. The BSW is also connected to the hardware interface containing the CPU, RAM, clock and device interfaces, see Figure 8.1 [7].

![Figure 8.1: Flight hardware implementation of application and basic S/W](image)

For testing, a basic software simulator (BSS) is required at an early stage before the OBC platform is obtained, see Figure 8.2 [7]. This system should allow functional testing of the algorithms without actually deploying the glider in water. Therefore, the glider dynamics and sensor hardware need to be simulated and be used as input data (Sim Task in Figure 8.2). The BSS will allow testing all interfaces between BSW and ASW, the ASW functionality in open-loop configuration (e.g. to be used to analyse logged trajectory data), the ASW functionality in closed-loop configuration and provide a preliminary assessment of the OBC performance in terms of CPU occupancy.

In the BSS setup, the device interfaces are connected to the terminal, file system, keyboard i.e. according to the testing environment. The protocols for the target and simulation interfaces are the same although their implementation and usage are different in both environments. The most straightforward implementation of the BSS would be a library that could be linked to the ASW such that a non real time executable is obtained which can be run on a single Windows PC. The most important feature is that the ASW code does not require any modifications regardless of the hardware
and operating system it is running on, enabling an easy transition from simulated lab setup to real mission setup.

For the closed loop simulator implementation of the ASW and BSW, all sensors must also be simulated to provide input data, these emulators must send and receive data to the OBC exactly in the same format as the real sensors would so that the ASW should not be affected when switching from simulation to operational mode. One of the emulated sensors is presented in Chapter 9.

### 9 AHRS Sensor Emulator

The AHRS sensor will send its data to the OBC during operation to estimate the position of the glider and should be mounted as close as possible to the centre of buoyancy and far away from magnetic disturbances as possible in order to obtain the most accurate measurements due to the sensitivity of the internal magnetometer. In order to test control algorithms on the MOTH glider before it is in real operation, an AHRS emulator and OBC emulator have been programmed. To test the data communication ASW in the lab, simulated data needed to be collected and was used as an input to the sensor emulator which duplicate the functions of the real sensor. The simulated was converted so that it could be read by the on-board computer in the same format as the real sensor would transmit data, subsequently, if the emulated systems was replaced by the real glider systems the OBC should read and obtain data in the same way as when connected to the notebook/simulated system to enable an easy switch from simulation to operational set up, see Figure 9.1.

#### 9.1 AHRS

The AHRS consists of a three axis sensor system that provide attitude information and is the main source of information for the MOTH Glider’s GNC system. The key component of the AHRS is the internal measurement unit (IMU) that use a combination of accelerometers and gyroscopes to measure
the glider’s velocity, orientation and gravitational forces, it also consists of an addition of an on-board processing system i.e. extended estimation filter. The selected AHRS for the MOTH glider is the LORD Microstrain 3DM-GX4-25 which contain a dual on-board processor running an adaptive Kalman filter that provides attitude estimates and internal measurements for heading, roll, pitch, yaw, angular rates, angular accelerations, linear accelerations, magnetic field among other features. The 3DM-GX4-25 sensor has a mini RS-232 connector and a converter to USB to enable serial communication between computers also without a physical RS232 port, see Figure 9.2. To emulate a successful communication between the AHRS and the OBC, the first step was to initialise the AHRS parameters, e.g. baud rate, sample rate, initial heading and attitude. The OBC then has to wait for a reply for each command it sends to the AHRS, once all initialisation commands have been replied the OBC can commence to ask for data to estimate the glider’s current position and orientation. Note that a small error in the attitude estimation will prolong the duration of the mission and can be reduced by increasing the sample frequency, the error tolerance depends on the mission. A technical specification for the
3DM-GX4-25 sensor with key data is presented in Table 9.1.

Table 9.1: 3DM-GX4-25 technical specification

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical attitude accuracy</td>
<td>±0.25° RMS roll and pitch, ±0.8° RMS heading</td>
</tr>
<tr>
<td>Typical attitude repeatability</td>
<td>0.3°</td>
</tr>
<tr>
<td>Attitude Resolution</td>
<td>&lt;0.01°</td>
</tr>
<tr>
<td>Attitude heading range</td>
<td>360° about all three axes</td>
</tr>
<tr>
<td>Accelerometer range</td>
<td>±5g standard (±16g option)</td>
</tr>
<tr>
<td>Gyroscope range</td>
<td>±300°/sec standard (±75°/sec,±150°/sec options)</td>
</tr>
<tr>
<td>Features</td>
<td>Adaptive Kalman Filter with gyroscope and acceleration bias tracking, magnetometer hard and soft compensation tracking, vehicle dynamics mode selection, adaptive measurement noise enable/disable, selectable internal or external heading sources, full world magnetic model.</td>
</tr>
<tr>
<td>Estimation Filter update rate</td>
<td>500 HZ</td>
</tr>
<tr>
<td>IMU Data output rate</td>
<td>1 to 1000 Hz</td>
</tr>
<tr>
<td>Interface</td>
<td>USB 2.0 and RS232</td>
</tr>
<tr>
<td>Power supply</td>
<td>+3.2 to +36 volts DC</td>
</tr>
<tr>
<td>Shock limit</td>
<td>500g</td>
</tr>
<tr>
<td>Dimensions</td>
<td>36×24.4×11.1 mm</td>
</tr>
<tr>
<td>Weight</td>
<td>16.5 grams</td>
</tr>
<tr>
<td>Data Outputs</td>
<td>Acceleration, angular rate, magnetic, field, delta Theta, delta Velocity, GPS Time, filter status, attitude estimates (Euler angles, orientation matrix, quaternion), Attitude uncertainties, gravity-free linear acceleration, bias-compensated angular rate.</td>
</tr>
</tbody>
</table>

9.2 Data Communication Protocol

The AHRS protocol is packet based, i.e. all commands, replies and data are sent and received as fields in a message packet that have a descriptor type field based on their contents to distinguish if a package contains commands, replies or IMU data. The IMU and Kalman filter data are grouped into a single package with a common GPS formatted timestamp to allow a precise common time base for all data, communication diagram presented in Figure 9.3 [8].

The chosen IMU data set is a scaled gyroscope vector and a delta velocity vector and the Kalman filter is used to estimate the Euler angles based on a linear dynamic system discretised in the time domain, modelled on a Markov chain built on linear operator, and resides on a separate processor that must derive its time information from the IMU data.

The sensors and instruments are connect to the on-board computer through a RS232 serial interface,
as seen in the electrical architecture, and it is possible to communicate with any device that has a serial
interface directly from Matlab. The AHRS commands and data are divided into four command and
two data sets corresponding to the inertial architecture of the device, see Table 9.2.

Table 9.2: Command and data overview

<table>
<thead>
<tr>
<th>Command/data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base commands</td>
<td>Ping, Idle, Resume, Get ID Strings</td>
</tr>
<tr>
<td>3D Motion commands</td>
<td>Poll IMU Data, Poll GPS Data, etc.</td>
</tr>
<tr>
<td>Estimation Filter commands</td>
<td>Reset Estimation Filter</td>
</tr>
<tr>
<td>System commands</td>
<td>Reset Estimation Filter</td>
</tr>
<tr>
<td>IMU data</td>
<td>Acceleration vector, gyro vector, Euler angles, etc.</td>
</tr>
<tr>
<td>Estimation Filter data</td>
<td>Attitude, acceleration estimation, etc.</td>
</tr>
</tbody>
</table>

The 3D motion commands are specific to the MicroStrain inertial product line and the estimation
filter and system commands are specific to the MicroStrain navigation and advances AHRS devices.

The MicroStrain Matlab integration package (MIP) structure consists of a four byte header and a
two byte checksum footer, see Figure 9.4 [8]. The header consists of "sync" bytes, these are the same
for every MIP packet and used simply to identify the start of the package. The header also consists of
a descriptor set byte which are grouped into different sets, in this case the 0x80 value identifies this
package as an AHRS data descriptor set, and a payload length byte which specifies the length of the
packet payload. The packet payload may contain one or more fields and therefore the payload length
byte represents the sum of the lengths of all the fields in the packet payload. The checksum serves as
a unique identifier of the data and consists of two byte fletcher checksum of all the bytes in the packet
and must thus be calculated for each data string. If the data change then so will the checksum, making
it easy to verify if data has been sent accurately according to the predefined commands and replies as
when these match with the received data there is a high degree of confidence that the data was sent
correctly.

The packet payload section, see Figure 9.5 [8], contains field length byte which represents a count of
all the bytes in the field including the descriptor byte and field data. A field descriptor byte identifies
the contents of the field data, which in this example represents the floating point magnetometer vector
(0x06) from the AHRS data set (0x80). The length of the data is always the field length data minus two
Design and Implementation of a Glider Control System

Figure 9.4: Header & Checksum Overview

(for field length byte and field descriptor byte) so this data is 12 bytes long (0E=14, 14-2=12). These commands are written in hexadecimal code, see Table 9.3 for conversion, and the payload length byte in this example is "0E" which is hexadecimal code for 14. In Matlab, to emulate the real system, the AHRS sensor should also send and received data in hexadecimal code for which each number represents half of a byte i.e. four bits.

Figure 9.5: Packet Payload Overview

To communicate with the AHRS the basic command sequence begins with the host, that being the OBC, sending a command packet to the sensor, the sensor should then send a reply packet back to the OBC, these series of command and reply pairs are called a setup. The glider will operate with a continuous sequence\(^5\) and to reduce the amount of streaming data the device will be set in idle state during initialisation, which is referred to as step 1 in the protocol setup/initialisation sequence. In step 1, a command to put the device in idle state, i.e. disabling IMU and estimation filter data streams, has to be sent, see Figure 9.6 [8].

Figure 9.6: Set to Idle command: 7565 0102 0202 E1C7

The reply to this command is ACK or NACK for acknowledge or not acknowledged. Step 2 is to configure the IMU data stream format, meaning in which order different data should be received (i.e.

---

\(^{5}\)A polling data programme has later been requested and developed by modifying the previous continuous stream programme, see section 9.3.2
first Euler angles, then delta velocity, angular rate, etc.). Step 3 is to configure the estimation filter data stream format which requests the estimated latitude, longitude, height (LLH) position, estimated north east down (NED) velocity, estimated orientation in quaternion form, and filter status at 100 Hz. The fourth step is to save the IMU and estimation filter MIP message format, the command and replies for these steps are all described in detail the 3DM-GX4-25 Protocol available by contacting LORD Corporation6.

In the protocol , the second and third step for the continuous data command sequence covers an example case interested in a scaled gyroscope, scaled accelerometer and GPS correlation timestamp information at 1000 Hz, the "7565 0C0D 0D08 0103 0400 0105 00011200 012A 35" command for the IMU and the "7565 0C10100A 0104 0100 0502 0005 0300 0510 00053F31" command for the estimation filter therefore have to be altered. The scaled gyro vector used in the command example can be kept for the angular rate, however the rest will be replaced with the command for estimated Euler angles (0x82, 0x05) for roll, pitch, yaw and the command (0x80, 0x08) for delta velocity vector according to the attached system commands, also found in 3DM-GX4-25 Protocol. This gave the new command "7565 0C0A 0A08 0102 0500 6408 0064 DA71" for the IMU (with scaled gyro and delta velocity vector both setting a sample rate of 10 Hz) and "7565 0C07 070A 0101 0500 3237 54" for the estimation filter

---

6LORD Corporation MicroStrain Sensing Systems, www.microstrain.com sensing_support@LORD.com, +1 802-862-6629
Design and Implementation of a Glider Control System

with Euler angles.

It is possible to obtain data from the AHRS in two different modes: continuous data, i.e. enabling continuous data streams to run, or polling data, i.e. asking for a measurement with a chosen frequency. Initially a continuous stream programme was developed, however, upon request by the department the continuous programme was altered to create a pulling data programme. The first four steps are identical for both programmes, however, after the commands and replies will differ for the two modes. All hexadecimal strings needed for the MOTH glider’s AHRS and OBC emulators are presented in Table 9.4 for continuous data (incl. step 1-4) and Table 9.5 for polling data.

Table 9.4: AHRS commands and replies for continuous stream

<table>
<thead>
<tr>
<th>Command</th>
<th>Command Value</th>
<th>Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle Mode</td>
<td>756501020202E1C7</td>
<td>7565010404F10200D66C</td>
</tr>
<tr>
<td>IMU data-stream format</td>
<td>75650C0A0A080102050064080064DA71</td>
<td>75650C0404F10800E7BA</td>
</tr>
<tr>
<td>EF data-stream format</td>
<td>75650C07070A01010500323754</td>
<td>75650C0404F10A00E9BE</td>
</tr>
<tr>
<td>Save IMU &amp; EF format</td>
<td>75650C0804080300040A030000E31</td>
<td>75650C0804F1080004F10A00EA71</td>
</tr>
<tr>
<td>Enable IMU &amp; EF data streams</td>
<td>75650C0A0511011051101030124CC</td>
<td>75650C0804F1110004F11100FAB5</td>
</tr>
<tr>
<td>Disable IMU &amp; EF data streams</td>
<td>75650C0A05110100051101030022C5</td>
<td>75650C0804F1110004F11100FAB5</td>
</tr>
</tbody>
</table>

As seen in Table 9.4, the fifth step for the continuous stream is to enable the IMU and estimation filter data-streams which triggers data to start being sent until the Disable Continuous Stream command is sent (change ON field data command to OFF). As the commands for idle, IMU format, EF format and Save are the same for both modes these are redundant to present in Table 9.5, here instead of enable and disable streams there are polling data commands with the option of sending commands with or without waiting for a reply. In the pulling data programme the OBC continuous to send command according to the set sample frequency until the user push stop.

Table 9.5: AHRS commands and replies for polling data

<table>
<thead>
<tr>
<th>Command</th>
<th>Command Value</th>
<th>Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poll IMU Data</td>
<td>75650C0404010000EFDA</td>
<td>75650C0404010000EFDA</td>
</tr>
<tr>
<td>Poll IMU Data no reply (optional)</td>
<td>75650C0404010010F0DC</td>
<td></td>
</tr>
<tr>
<td>Poll EF Data</td>
<td>75650C0404030000F1E0</td>
<td>75650C0404F103000E2B0</td>
</tr>
<tr>
<td>Poll EF Data no reply (optional)</td>
<td>75650C0404030100F2E2</td>
<td></td>
</tr>
</tbody>
</table>

After the commands and replies have been defined according to the protocol it was time to start working on the Matlab simulation.
9.3 AHRS and OBC emulators

A six degrees of freedom model of the glider was developed and programmed in Matlab by Uwe Soppa, from this it was possible to obtain the requested AHRS input data: Euler angles (pitch, roll, yaw), angular rates, depth and depth rate, see Table 9.6. In the Matlab programme, two additional columns were added for delta velocity x and delta velocity y in the data sheet as these are generated by the real AHRS so that the order and quantity of data will be read in the same format as the real AHRS, however, as the data in these two added columns could not be computed from the data simulation these were all put to zero.

Table 9.6: AHRS input data generated by 6DOF model simulator

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.7854</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>-0.0000</td>
<td>0.7854</td>
<td>0</td>
<td>-0.0010</td>
<td>0</td>
<td>0.0438</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>-0.0002</td>
<td>0.7854</td>
<td>0</td>
<td>-0.0029</td>
<td>0</td>
<td>0.0644</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>-0.0007</td>
<td>0.7854</td>
<td>0</td>
<td>-0.0055</td>
<td>0</td>
<td>0.0622</td>
</tr>
<tr>
<td>14999</td>
<td>-0.1072</td>
<td>-0.2818</td>
<td>-0.9274</td>
<td>0.0326</td>
<td>-0.0011</td>
<td>-0.0021</td>
<td>0.3894</td>
</tr>
<tr>
<td>15000</td>
<td>-0.1039</td>
<td>-0.2819</td>
<td>-0.9277</td>
<td>0.0319</td>
<td>-0.0011</td>
<td>-0.0022</td>
<td>0.3898</td>
</tr>
</tbody>
</table>

After the input data was obtained the baud rate was calculated according to

\[ n(k \times f_{mr}) + n \sum S_f \times f_{dr} \]  

where \( S_f \) is the size of data field in bytes, \( f_{dr} \) is the field data rate in Hz, \( f_{mr} \) is the maximum data rate in Hz, \( n \) is the size of UART word (10 bits) and \( k \) is the size of MIP wrapper (6 bytes). This is of course correlated with the sample rate which is how often the sensor collects data. The delta velocity, Euler angles and gyro are initially set to the same sample rate, 10Hz, subsequently \( f_{dr} = f_{mr} = 10 \). The size of the data field, \( S_f \) are 16 [Euler angles: float, float, float, U16, field length, data descriptor i.e. 4+4+4+2+1+1], 14 [Gyro: float, float, float, field length, data descriptor i.e. 4+4+4+1+1] and 14 [Delta velocity: float, float, float, field length, data descriptor i.e. 4+4+4+1+1]. With these values, the baud rate equation gives the baud rate 5000 bits/s. The supported baud rates are 9600, 19200, 115200 (default for AHRS sensor), 230400, 460800, 921600 thus the baud rate is set to 9600, i.e. closest to the calculated result. Note that AHRS and OBC must have the same baud rate to be synchronised.

The next step was to select the communication port, which can be opened in Matlab using the \texttt{fopen} command, to write binary data to the port the \texttt{fwrite} command is used and to read from the port the \texttt{fread} command is used. Once communication between the OBC emulator and the virtual serial port has been established the next step is to send all commands according to the AHRS protocol, see Table 9.4, to validate the data communication.
For the emulated simulations, two computers are needed, one taking the role as the OBC and the other as the AHRS. These will be connected with an USB-RS232 cable, as shown in Figure 9.2, as the real system.

Two programmes were as mentioned developed, the first one for continuous data streaming where the AHRS continuously reads and sends data until the OBC sends a disable command. The second program is polling data which is programmed to ask for one measurement and first then the AHRS commence to read this one measurement and the duration between sending commands, i.e. the sample frequency, is set by the user.

### 9.3.1 Continuous data stream programme

Before the RS232 cable was obtained, the sensor emulator and the OBC was initialised using a virtual serial port, where data was sent and received as a series of bytes. A terminator was set to mark the end of a series of bytes that constitute a message, also a buffer was and remains in use to store the incoming data until it is read. The buffer can be seen as a list with a finite length, the length can be selected. Once the buffer is full and new data comes in, the oldest data, i.e. on top of the list, is discarded and all other entities moves up to make room for the new data in the bottom of the list. Once data has been disregarded it can never be restored thus the baud rate should preferably be selected so that data is never lost.

The developed continuous data streaming programme, available upon request\(^7\), consists of several scripts with different extensions described below:

- **ahrs_main.m**, AHRS emulator main file, initialise sensor, continuously reads data and sends this to the OBC based on the sample rate.
- **obc_main.m**, OBC emulator main file, enables and disables the AHRS and continuously reads data sent from the AHRS.
- **obc_init.m**, OBC emulator initiation file includes step 1-5, needs to be run before obc_main.
- **converter.m**, convert numeric data to hexadecimal data and format the string.
- **data.m**, contains simulated data to test the emulated system on.
- **floatconv.m**, presenting the numeric data as floating
- **ftest.m**, calculates the Fletcher Checksum for the hexadecimal strings.
- **hex2num1.m**, convert hexadecimal data to numeric data using an already preprogrammed function hex2num with minor changes to maintain single 32 bit data unlike the original file that gave 64 bit data.

\(^7\)Contact: Hannah Lindberg, hannah.lindberg@airbus.com, +4917683324248
In the AHRS main file and the OBC init file, the initiation commands and replies were defined as presented in Table 9.4 according to the AHRS protocol for idle, baud rate, set IMU and EF format, save format, enable stream and in the AHRS main and OBC main files also of course the commands and replies for enable and disable stream to continue to run the programme after initiation. The AHRS main file contains two while loops that continuously reads and sends commands and replies to the OBC main and init files. The first loop runs until the AHRS is successfully initialised and the second, after the comport and buffer are checked, continuous to run until it has received a disable command.

9.3.2 Polling data programme

For the polling data, the function files: converter, floatconv, ftest and hex2num1 remained unaltered whilst the AHRS main and OBC init and main files were modified to polling data with commands and replies defined according to Table 9.5. The OBC emulator first initialise communication with the AHRS emulator by establishing the port number and sample rate which has now been programmed to be controlled by the user from a pop-up window, see Figure 9.7. This is run by the comport.m and samplerate.m switch statement functions which are waiting for the argument by the user and then set a value based on that.

![OBC Emulator Interface](image)

Figure 9.7: OBC Emulator Interface

The OBC init file contains the first four steps of the initiation command and replies are run in a while loop as for the continuous case that runs until successfully completed, the enable stream is however not used here instead the AHRS now only polls a measurement once triggered by the user from the OBC main file, how often a data request command is sent from the OBC is decided by the sample rate set by the user in the popup window.
The developed polling programme, available upon request\(^8\), consists of several scripts with different extensions described below as well as the previous function files:

- `ahrs_mainp.m`, AHRS emulator main file, initialise sensor, reads commands from OBC and sends data upon request.

- `obc_mainp.m`, OBC emulator main file, user controls when to ask for data from this file. This file is using a built-in GUI Interface Matlab tool which has been modified to set specific variables and values.

- `obc_initp.m`, initialise all the parameters of the AHRS, preparing it for polling data command. Its arguments are port number and baud rate in order to establish the communication, this function will return a boolean variable 1 = success and 0 = failure.

- `comport.m`, contains a switch control function created to return the value of the selected comport in the pop-up window.

- `samplerate.m`, contains a switch control function created to return the value of the selected sample rate in the pop-up window.

- `setformat.m`, contains a function to pick specific data

The polling programme use the same data file as the continuous programme however the setformat file is new and defines which specific data should be pulled based on a set format function: (`source`, `number of variables`, `description`, `sample rate`), thus it is possible to define to pull e.g. IMU data, which variables and how often.

The timing between the commands are sent from the OBC is controlled by a ticktock timer which gives better control of the time between readings compared to the continuous streaming and the user can control and check this time which was not previously possible. The maximum sample rate is currently 9Hz as it takes 0.01 seconds to read the data in the while loop for sending and receiving commands.

10 Conclusion

The set up objectives have been met and the variety of assignments within this thesis project have together given a greater understanding of the break down of system engineering projects in the industry. This project has also presented fruitful challenges both technical and in regards to the added complexity of system design when several partners from different institutions are involved requesting many iterations.

\(^8\)Contact: Hannah Lindberg, hannah.lindberg@airbus.com, +4917683324248
The thesis project was initialised with a literature study of glider history and hydrodynamics to get acquainted with the field and the glider system requirements were reviewed and updated. An overview of the glider system architecture was created using SysML and MS Visio, where all main system components were identified, and an overview of the OBC system tasks was created with the same softwares. Following, a use case study using SysML was carried out to identify the capabilities of the glider system and the responsibilities of different actors, e.g. the control station, and the possible tasks to be carried out in different scenarios e.g. during recovery, operation or initialisation.

In MS Visio, an electrical architecture was created to determine the bus systems/interfaces connected to the OBC and CAN bus was selected as the primary interface and CAN bus - RS232 converters where added where needed. Remote terminal units were implemented to enable easy swops of scientific measurement instruments and serve as a border between the static and flexible system. Linux Debian was select for the chosen ARM based embedded Linux computer, both through trade-off analysis and a system state machine was designed to define possible states, events, actions and transitions to help to identify possible control issues.

A pitch controller was designed using Matlab and Simulink to simulate the gliders longitudinal flightpath, for which several parameter assumptions had to be made because the real parameters are yet to be announced as the MOTH is in an early stage of the development. This parameter uncertainty created difficulties with stabilising the model once the glider reached transition points. Increasing the control gains, $K_P$ and $K_D$, made the flap deflection angle reach their maximum deflection faster, altought unable to compensate for the pitch error which was also the case with decreasing the gains which merely prolong the time until the flaps reach maximum deflection and gave a higher error.

Also in Matlab, an AHRS sensor emulator and an OBC emulator were programmed according to the AHRS data communications protocol to enable communication between the AHRS and OBC for simulations in the laboratory environment. Both a continuous stream and a polling data programme were designed and tested.

The contributed work within the scope of this thesis will continue to be used within the MOTH project in the upcoming steps which are, in accordance to the V-model, to carry out integration and testing. This will be done with an underwater test using a launch pad to identify the parameters and then the pitch controller will be used and updated with new parameter values. The sensor emulator will also be used, along with other emulated systems, in the simulation and verification tests carried out in the laboratory environment. For the simulation tests there is a robotic arm, already available in the laboratory, that will be used to carry the glider.

After completing the verification and validation, a demo mission will be carried out in a swimming pool and the first mission is planned in 2017.
11 Discussion

Given the current interest in exploring the oceans and the technical development within the field, it is promising that the MOTH glider could just be the first of a series of fully autonomous underwater gliders benefiting from the synergy effects found within ROBEX. The MOTH glider is aimed to reach a depth of 1000 meter and operate for 48 hours with a maximum velocity of 1 m/s, the next gliders with a higher velocity and durability to reach further and the bottom of the oceans would start to be explored. The durability mainly depends on the power consumption, thus with either less power consuming components or more durable batteries or a combination of these would in the future enable this progress.

The MOTH glider is under development within the ROBEX alliance consisting of 16 different partners, having several engineers involved, from both the space and deep sea industry, with different preferences can prolong the decision making and demands more iterations before a final architecture or design is agreed upon. Although the final system will have had the benefit to be reviewed from many different angles.

One design challenge with the glider is the limited accommodation and low weight preferences, otherwise it would be no problem to already instal more powerful batteries. The limited accommodation and weight are of course motivating to find lighter materials much like within the airplane industry. New materials are under discussion to be implemented both for the glider envelope as well as for the three internal tubes, the progress of the light structure material research also yield for a promising future of the underwater gliders.

Another challenge with underwater gliders is the limited communication. In the current state of the MOTH glider, the implementation of an acoustic transceiver is not yet certain, the glider has to go back up to surface to transmit and receive data via satellite communication. The glider has a built in memory and could therefore store data to avoid frequent ascends to the surface, which for shorter missions, should be sufficient. For longer missions at greater depths it could however be desirable to be able to communicate with the glider directly while it is still submerged which would be interesting to investigate as a future way forward.
12 Appendices

A Functional Requirements

The OBC shall provide the functions necessary to perform the following tasks:

- Execution of the Core Control Software including the following main functions:
  - Attitude/Depth control of the glider
  - Handling of on-board control
  - Navigation
  - Housekeeping
  - Commands
  - Power Control
  - Mode Management (eg. Safemode) Procedures (OBCP)
  - Failure, Detection, Isolation and Recovery (FDIR)

- On-board time synchronisation (supported by GPS if available)

- Acquisition of telemetry and events

- Acquisition of sensor data

- Time-stamped logging of events and sensor data

- Control of actuators

- Storage and retrieval of sensor and logging data in non-volatile memory

- Payload Management & Processing

The OBC shall control all active components of the Glider, i.e. navigation & status sensors, actuators (e.g. buoyancy S/S, emergency emersion S/S, communication S/S, PCDU) and payloads (scientific measurement instruments). The emergency signalling (e.g.LED) shall be independent from the OBC.

Physical

The whole controller including all interface components and support elements (excluding the housing) shall have a mass of less than XX g and a size of XX mm x XX mm x XX mm [TBC].

Operational

The activities in the autonomous operations shall be controlled by a pre-defined timeline that describes all required activities. The system shall have the following operational modes: Off, Checkout, Preparation, Standby, Operational, Start-up Failure, Operational Failure, Emergency, Maintenance, Recovery.
Design

The OBC shall consume a maximum power of 5 W [TBC] and provide the capability of hosting a real-time operating system. The OBC shall be able to handle a minimum of 10 instruments/sensors connected via serial interfaces. Each separated component of controller shall have mounting interface points allowing their accommodation inside the controller pressure container. The S/W architecture shall contain a separate process for the monitoring of the sensor and status data, to detect failures, to isolate them and the activation of recovery actions. This process shall have a priority which ensures an operation not disturbed by payload activities. It shall be possible to command the clock rate of the computer. This shall be possible by command received via TC and by command generated by the internal timeline. The status of the battery shall be monitored via a sensor. The systems reference time shall have a minimum resolution of 100 ms.

Time

In the emerged state, the OBC shall be able to synchronise its reference time to an external time source (GPS) and during checkout, the OBC shall be able to synchronise its reference time with the control station. The OBC shall be able to have a free-running reference time in case no external time source is available and shall distribute this reference time through the entire system. The internal control loop rate shall be 100ms. The time control application shall use UTC and all data that is written to the mass memory shall be time-stamped.

Communication - External

In the dived state, the OBC shall send a reduced set of telemetry and activate the receiver in predefined and adjustable intervals for a predefined and adjustable time to read commands. In the emerged state, the OBC shall receive commands continuously and shall, in its nominal state, send once an extended set of telemetry, e.g. position, data etc. and in non-nominal state, the OBC shall only send its position periodically.

Communication - Internal

The OBC shall be able to read the sensor data periodically, with an adjustable reading interval for each sensor in the range from 100 ms to 1 h [TBC]. The OBC shall be able to send commands to the payloads asynchronously and receive status and data information from the payloads in a predefined and adjustable interval. The OBC shall be able to control and get status information (e.g. position) from the actuators.
Monitoring/Data

The OBC shall monitor the status of each sensor, payload and actuator in a predefined and adjustable interval and be able to read and write to persistent memory. The payload sensor data acquisition rate shall be adjustable for each sensor between 1s and 100ms. The on-board software shall store logging, selected housekeeping and payload data at defined periods in mass memory and it shall be possible to access data, e.g. log files, payload data, etc., stored on the mass memory from an external computer.

Modes/Operations

After an emersion operation, the system shall get its position using GPS and to connect to a global satellite network (e.g. Iridium) to transmit its position.

FDIR

The OBC shall provide a set of FDIR strategies for non-nominal behaviour that shall be selected based upon the occurred events, e.g. look-up table. The on-board software shall monitor each application for liveliness, in case of a non-responsive application, the system shall execute a corresponding FDIR strategy. After a system dropout the OBC shall reboot immediately autonomously and the on-board software shall protocol all detected anomalies to a log-file and have a boot counter. The OBC shall monitor the sensor data for being within adjustable operational limits. The OBC hardware shall provide a hardware watchdog which shall reset the OBC if not triggered on time. In case the energy reserve is too low, the on-board software shall trigger an emergency emersion, this applied for any non-nominal situation from which the system can not recover autonomously nor remotely.

Navigation

Based upon the input of the sensors, the OBC shall provide the means to navigate the glider along its planned trajectory. If a GPS signal is available, the on-board software shall calibrate its position. If no GPS signal is available, the on-board software shall calculate its own position using the last known GPS position and the navigational set of sensors.

Operating System

The OBC shall use a real time operating system. The OS shall be able to execute up to 10 [TBC] tasks simultaneously and the scheduling of the OS shall be time and priority controlled. It shall be possible to install new boot images and prior loading the boot image, the OBC shall be able
to detect a corruption of the boot image. The booter shall have at least two boot images, one nominal and one emergency backup image.

**Power Control**

The power control application shall be able to turn devices on/off according to the current mission phase.

**Interface**

For the demo-mission it is sufficient to establish interfaces to a pre-defined set of sensors only, it shall be possible to extent the interfaces to the sensors for later missions. The controller shall have an interface to the emergency ascent system. This interface shall provide the following functionalities: [TBD], status data from the emergency ascent system and activation of the emergency system. The interface between controller and emergency ascent system shall be [TBD]. Only I/F providing a digital I/O shall be used. In the emerged state, the glider shall communicate via the Iridium satellite, in the dived state, via acoustic means and during the checkout phase, via Ethernet with the control station.

**Environmental**

The glider should be able to operate in temperatures from -5 to +50 °C and shall be able to operate submerged in oil (with no extra pressurised compartment).
### B Device list

The selected, as of April 2015, devices to meet the functional requirements are listed in Table 12.1 with power and interface requirements listed.

<table>
<thead>
<tr>
<th>Device</th>
<th>Model</th>
<th>Supply</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sensors</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Acoustic Position System</td>
<td>GAPS</td>
<td>24-36 VCD</td>
<td>RS232, RS485, Ethernet</td>
</tr>
<tr>
<td>- AHRS</td>
<td>3DM-GX4-25</td>
<td>3.2-36 V</td>
<td>RS232, USB</td>
</tr>
<tr>
<td>- GPS</td>
<td>NovAtel - Software, FlexPack6</td>
<td>6-36 V</td>
<td>RS232, RS422, CAN- bus, USB, Ethernet</td>
</tr>
<tr>
<td>- Pressure sensor</td>
<td>minilPS</td>
<td>9-30 V (isolated)</td>
<td>RS232, RS485</td>
</tr>
<tr>
<td><strong>Payload</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- CTD-O</td>
<td>CTD48/CTD48M</td>
<td>7-15 V</td>
<td>RS232, RS485</td>
</tr>
<tr>
<td>- Current Meter</td>
<td>MAVS-3</td>
<td>13.5 VCD</td>
<td>RS232, RS485</td>
</tr>
<tr>
<td>- Flourescence sensor</td>
<td>Microflu-blue</td>
<td>5-15 V</td>
<td>RS232</td>
</tr>
<tr>
<td>- Led Flasher</td>
<td>XMF-11K</td>
<td>6 AA alkaline batteries</td>
<td>Microprocessor Controller</td>
</tr>
<tr>
<td>- Video System</td>
<td>Insite Tritech Nova</td>
<td>15 V</td>
<td></td>
</tr>
<tr>
<td><strong>Communication</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- L-Band Satellite Transceiver</td>
<td>Iridium 9522B</td>
<td>5,12,24 VCD Nominal</td>
<td>RS232</td>
</tr>
<tr>
<td>- Serial Wifi Device Server</td>
<td>SW5501/ SW5502</td>
<td>9-48 VCD</td>
<td>RS232, RS422, RS485</td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- OBC</td>
<td>GX-6300</td>
<td>8/40 V</td>
<td>RS232, Ethernet</td>
</tr>
</tbody>
</table>
C Glider Dimensions
<table>
<thead>
<tr>
<th>Brand Name</th>
<th>Pin1(Pin2) (Br.)</th>
<th>P1</th>
<th>Board/Board Stack</th>
<th>P1</th>
<th>T5-7670</th>
<th>P2</th>
<th>T5-7900</th>
<th>P2</th>
<th>T5-7200</th>
<th>P2</th>
<th>T5-3000</th>
<th>P2</th>
<th>com-pa-G966</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>no</td>
<td>1</td>
<td>200 MHz (A96)</td>
<td>1</td>
<td>200 MHz (A96)</td>
<td>1</td>
<td>200 MHz (A96)</td>
<td>1</td>
<td>200 MHz (A96)</td>
<td>1</td>
<td>200 MHz (A96)</td>
<td>1</td>
<td>200 MHz (A96)</td>
<td>1</td>
</tr>
<tr>
<td>RTOS</td>
<td>yes</td>
<td>4</td>
<td>512 MB</td>
<td>4</td>
<td>512 MB</td>
<td>4</td>
<td>512 MB</td>
<td>4</td>
<td>512 MB</td>
<td>4</td>
<td>512 MB</td>
<td>4</td>
<td>512 MB</td>
<td>4</td>
</tr>
<tr>
<td>CPU</td>
<td>no</td>
<td>1</td>
<td>100 MHz (A96)</td>
<td>1</td>
<td>100 MHz (A96)</td>
<td>1</td>
<td>100 MHz (A96)</td>
<td>1</td>
<td>100 MHz (A96)</td>
<td>1</td>
<td>100 MHz (A96)</td>
<td>1</td>
<td>100 MHz (A96)</td>
<td>1</td>
</tr>
<tr>
<td>Memory</td>
<td>no (only via external IO)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
</tr>
<tr>
<td>Ethernet</td>
<td>yes (10100)</td>
<td>4</td>
<td>yes (10100)</td>
<td>4</td>
<td>yes (10100)</td>
<td>4</td>
<td>yes (10100)</td>
<td>4</td>
<td>yes (10100)</td>
<td>4</td>
<td>yes (10100)</td>
<td>4</td>
<td>yes (10100)</td>
<td>4</td>
</tr>
<tr>
<td>CAN</td>
<td>no (only via external board)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
</tr>
<tr>
<td>RS485</td>
<td>no (only via external board)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
<td>yes (10100)</td>
<td>2</td>
</tr>
<tr>
<td>FPGA</td>
<td>no</td>
<td>3</td>
<td>no</td>
<td>3</td>
<td>no</td>
<td>3</td>
<td>no</td>
<td>3</td>
<td>no</td>
<td>3</td>
<td>no</td>
<td>3</td>
<td>no</td>
<td>3</td>
</tr>
<tr>
<td>Additional Features</td>
<td>- Dual Core VideoCorr inIEEE</td>
<td>5</td>
<td>yes (10100)</td>
<td>5</td>
<td>yes (10100)</td>
<td>5</td>
<td>yes (10100)</td>
<td>5</td>
<td>yes (10100)</td>
<td>5</td>
<td>yes (10100)</td>
<td>5</td>
<td>yes (10100)</td>
<td>5</td>
</tr>
<tr>
<td>Calculated Points</td>
<td>- Watchdog and Standby</td>
<td>2.9</td>
<td>0.29</td>
<td>0.29</td>
<td>0.29</td>
<td>0.29</td>
<td>0.29</td>
<td>0.29</td>
<td>0.29</td>
<td>0.29</td>
<td>0.29</td>
<td>0.29</td>
<td>0.29</td>
<td>0.29</td>
</tr>
</tbody>
</table>

**D OBC Trade-off**

**Design and Implementation of a Glider Control System**

- **Power**
  - 100 MHz (A96)
  - 200 MHz (A96)

- **RTOS**
  - Yes

- **CPU**
  - No

- **Memory**
  - 512 MB

- **Ethernet**
  - Yes (10100)

- **CAN**
  - No (only via external board)

- **RS485**
  - No (only via external board)

- **FPGA**
  - No

- **Additional Features**
  - Dual Core VideoCorr inIEEE

- **Calculated Points**
  - Watchdog and Standby

---

**Notes:**

- **Pin1(Pin2) (Br.):** Various configurations are provided for each feature.
- **Board/Board Stack:** Different board configurations are listed.
- **T5-7670, T5-7900, T5-7200, T5-3000:** Various performance metrics are compared.
- **Com-pa-G966:** A comparison metric is shown.
- **Power & Memory:** Various power and memory configurations are compared.
E  OS Trade-off
References


   aScripps Institution of Oceanography, La Jolla, CA 92093-0230, USA,
   bSchool of Oceanography, University of Washington, Seattle, WA 98195-5351, USA and
   cWebb Research Corporation, Falmouth, MA 02536, USA


