



THE AUSTRALIAN NATIONAL UNIVERSITY

# Inertial Based Control on the Kambara Project

David Gregory Biddle  
3221429

23 June 2003

A thesis submitted in part fulfillment of the degree of Bachelor of  
Engineering at the Australian National University

Supervisor: Dr Uwe Zimmer

Except where otherwise indicated, this thesis is my own original work.

David Biddle  
23 June 2003

---

# Abstract

---

This thesis presents details of the continuing efforts to develop a reliable navigation system for an autonomous underwater vehicle (AUV) at the Research School of Information Systems and Engineering at the ANU. In particular, work on the development of an inertial control system for the AUV, Kambara is described. The design of the inertial control system is based on a detailed system model describing the hydrodynamic motion of the vehicle. It incorporates four inertial sensors taking 10 measurements on the current state of the AUV. A real time analysis is performed to verify that the AUV will rapidly respond to feedback commands in the real world environment. Testing is performed on the control design and the real time system. Results are presented which theoretically demonstrate that the controller will stabilise the acceleration and velocity and reach the target position of the AUV in the six degrees of freedom underwater environment.

---

# Acknowledgements

---

My time at RSISE, working on the Kambara project has been a great opportunity. Everyone within the department and within the project team has been welcoming and provided me with useful assistance. The Kambara project has been a great Systems Engineering project to help utilise the knowledge I have acquired during my studies.

In particular, I would like to thank the following people:

- My supervisor, Dr Uwe Zimmer, for continuing to support me through this project whilst teaching overseas in Germany;
- Christfried Webers, who has effectively been a supervisor, assisting me throughout my project. A special thank you for the hard work you completed on the low level code for Kambara;
- My parents for their invaluable support and encouragement during the past few months;
- Rebecca Newman, who has continued to encourage and support me with her prayers and love; and
- Alexander Bahr, who has continued to encourage me to meet my required deadlines around the office.

Thank you to everyone else at RSISE for helping me with my project. I would like to thank my fellow Engineering students for providing encouragement to me throughout this period. Most importantly I would like to thank Jesus Christ, who has been with me on every step of this project.

---

# Contents

---

<b>Abstract</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Project Background . . . . .	1
1.2 Kambara System Description . . . . .	2
1.3 Other AUV Research Projects . . . . .	4
1.4 Contributions . . . . .	4
1.5 Project Objectives . . . . .	5
1.6 Outline . . . . .	6
<b>2 System Model</b>	<b>7</b>
2.1 Reference Frame . . . . .	7
2.2 Attitude Representation . . . . .	8
2.2.1 Euler Angles . . . . .	8
2.2.2 Quaternions . . . . .	9
2.2.3 Transformation from Euler Angles to Quaternions . . . . .	11
2.3 Vector State Representation . . . . .	11
2.3.1 Position State Vector . . . . .	11
2.3.2 Velocity State Vector . . . . .	12
2.3.3 Thrust State Vector . . . . .	12
2.4 Vehicle State Representation . . . . .	13
2.4.1 Thruster Forces and Torques . . . . .	13
2.4.2 Mass and Inertia Matrix . . . . .	14
2.4.3 Hydrodynamic Added Mass Matrix . . . . .	15
2.4.4 Hydrodynamic Added Mass Coriolis-Like Matrix . . . . .	15
2.4.5 Hydrodynamic Damping Matrix . . . . .	15
2.4.6 Gravitational and Buoyancy Forces Vector . . . . .	16
2.4.7 Thrust Mapping Matrix . . . . .	16
2.5 Summary . . . . .	17

---

<b>3</b>	<b>Sensor Suite</b>	<b>18</b>
3.1	Digital Compass - TCM2-50 . . . . .	19
3.2	360° Absolute Inclinometers . . . . .	20
3.3	MotionPak . . . . .	21
3.4	Pressure Sensor . . . . .	22
3.5	Sensor Communications . . . . .	23
<b>4</b>	<b>Kambara Controller</b>	<b>25</b>
4.1	Controller Derivation . . . . .	25
4.2	Reference Values . . . . .	27
4.3	Control Parameter Design . . . . .	29
<b>5</b>	<b>Thruster Controller</b>	<b>31</b>
5.1	Thrusters . . . . .	31
5.2	Accelus Amplifiers . . . . .	31
5.3	Thruster Control System . . . . .	32
<b>6</b>	<b>Real Time Analysis</b>	<b>34</b>
6.1	Operating System and Programming Language . . . . .	34
6.2	Real Time System Design . . . . .	35
<b>7</b>	<b>System Testing</b>	<b>37</b>
7.1	Controller Response . . . . .	37
7.1.1	Acceleration Testing . . . . .	37
7.1.2	Angular Velocity Testing . . . . .	38
7.1.3	Position Testing . . . . .	39
7.1.4	Overall Effects . . . . .	41
7.2	Real Time Integration . . . . .	41
7.3	Further Testing . . . . .	41
7.4	Possible Errors in the Controller . . . . .	42
<b>8</b>	<b>Conclusion</b>	<b>47</b>
	<b>Bibliography</b>	<b>49</b>
<b>A</b>	<b>MotionPak Data Sheet</b>	<b>51</b>
<b>B</b>	<b>Thruster Output</b>	<b>52</b>
<b>C</b>	<b>Depth Sensor Testing</b>	<b>54</b>

---

<b>D</b>	<b>Ada Controller Code</b>	<b>56</b>
<b>E</b>	<b>Inclinometer Testing Code</b>	<b>69</b>
<b>F</b>	<b>CD-ROM directory Structure</b>	<b>73</b>

---

# List of Figures

---

1.1	Sedentary Fouling Serpulid Worm . . . . .	2
1.2	Kambara . . . . .	3
2.1	World and local Reference Frames . . . . .	8
2.2	Euler Angles . . . . .	8
2.3	Thruster Layout . . . . .	14
3.1	Kambara Sensor Suite . . . . .	18
3.2	Compass Sensor(left) & Inclinometers(right) . . . . .	20
3.3	MotionPak(left) & Pressure Sensor(right) . . . . .	22
3.4	Communication of Sensor and Output Data . . . . .	24
4.1	Feedback Control System for Kambara . . . . .	29
4.2	System Step Response: $k_2 = 10, k_1 = 1$ . . . . .	30
4.3	System Step Response: $k_2 = 5, k_1 = 10$ . . . . .	30
4.4	System Step Response: $k_2 = 1, k_1 = 2$ . . . . .	30
4.5	System Step Response: $k_2 = 0.1, k_1 = 10$ . . . . .	30
5.1	Accelus Amplifier . . . . .	32
5.2	Thruster Step Response: $C_p = 100, C_i = 50$ . . . . .	33
5.3	Thruster Step Response: $C_p = 1500, C_i = 0$ . . . . .	33
5.4	Thruster Step Response: $C_p = 3000, C_i = 500$ . . . . .	33
5.5	Thruster Step Response: $C_p = 1500, C_i = 200$ . . . . .	33
5.6	Thruster Step Response: $C_p = 2000, C_i = 500$ . . . . .	33
6.1	Real Time System Data Flow . . . . .	35
7.1	Acceleration in X direction . . . . .	38
7.2	Angular Velocity in X direction . . . . .	39
7.3	Displacement in Z direction . . . . .	40
7.4	Timeline, Representing Access to the Protected Sensor Object . . . . .	42
7.5	Acceleration in Z direction . . . . .	43
7.6	Angular Velocity in Y direction . . . . .	44



---

7.7	Angular Velocity in Z direction . . . . .	44
7.8	Angular Displacement in X direction . . . . .	45
7.9	Angular Displacement in Y direction Showing Singularities	45
7.10	Angular Displacement in Z Direction . . . . .	46
7.11	Non Singularity Angular Displacement in Y direction . .	46
A.1	Manufacturer's data sheet for the MotionPak . . . . .	51
B.1	Minnkota Thruster Output Results . . . . .	53
C.1	Pressure Sensor Output . . . . .	54
C.2	Sensitivity Testing of Pressure Sensor . . . . .	55

---

# Introduction

---

## 1.1 Project Background

Throughout the history of mankind, humans have continued to explore and discover the world we inhabit. The ocean covers 70 % of the Earth's surface and is still relatively unknown, holding vast biological and mineralogical resources which are of great interest to mankind. Currently we do not know enough about our oceans to fully utilise these resources. The ocean must be investigated and understood so that this environment can be sustainably developed and effectively protected. [18]

The ocean is an extremely harsh environment characterised by complex physical and chemical variability at its surface and by depth dependant temperature, light and pressure gradients. Sea water is highly corrosive of metals and, with increasing depth, pressure effects pose fundamental challenges. The ocean floor is erratically contoured and contains a vast array of biological life. Some marine organisms may threaten the ever increasing shipping operations that occur world wide. For Example, the sedentary fouling serpulid worm, *Hydroides sanctaecrucis* (Figure 1.1) is an invasive pest capable of damaging the hulls of ships and many other submerged structures [13]. This marine pest, can form extensive and dangerous reefs on wharves, pontoons and slow moving vessels. Research currently being undertaken by the CSIRO into this invasive marine pest relies upon underwater observation of these pests in order to access their rate of colonisation and spread.

The underwater environment is harsh to humans with no useable oxygen, high pressure levels and its inherent resistance to movements. In order to investigate this environment, underwater explorations have typically been carried out by humans aided by various apparatus. These apparatus help humans to freely move and record data in the harsh underwater environments. Operations such as collecting marine samples, laying underwater cables and inspection and construction of underwater structures are examples of current endeavors. Remote operated



**Figure 1.1: Sedentary Fouling Serpulid Worm**

vehicles (ROV) and unmanned underwater vehicles (UUVs) have become an important tool for facilitating the performance of these tasks. UUVs are typically remote controlled from the surface and have been used in a wide range of operations including underwater research, deep water diver support, sea floor surveys and underwater inspection of pipes and structures.

A significant amount of the work completed by UUVs involves long duration and repetitive tasks. A degree of autonomy is desirable in these cases to remove the dependencies from the remote user. In order to address this issue research is currently being directed into Autonomous Underwater Vehicles (AUVs). A number of research institutions and Universities around the world are undertaking various AUV projects.

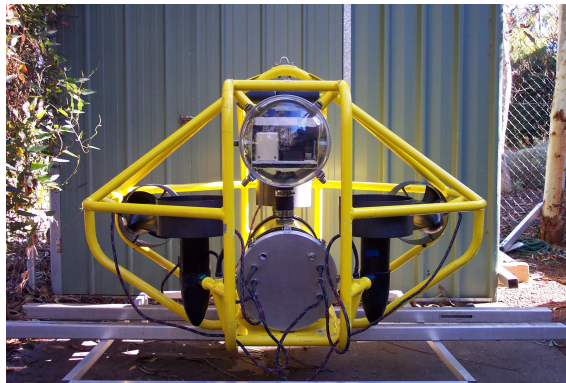
The Robotic Systems Laboratory (RSL), located within the Research School of Information Systems and Engineering (RSISE) at the Australian National University (ANU), is researching possible uses of AUVs. Currently, the RSL is undertaking two projects, the Kambara project and the Kambrini project. The Kambara project is currently examining the underwater navigation of the AUV using sonar technology, while the Kambrini project involves research into a swarm of small AUVs that are continually transferring data to each other.

## 1.2 Kambara System Description

Kambara is an AUV developed at the Robotics Systems Lab. The word Kambara originates from an Aboriginal word meaning crocodile. Kambara is an aluminium open frame low cost AUV, that is used as a basis for underwater research. Kambara incorporates five thrusters and two cylindrical enclosures mounted on an aluminium cylindrical frame. The frame is 1.2m long, 1.5m wide and has a

height of 0.9m. The assembled vehicle displaces approximately 110 liters of water. The mass of the frame, enclosures, and thrusters is approximately 68 kg in air.

The five thrusters mounted in two planes, enable movement in five degrees of freedom, namely roll, pitch, yaw, heave, and surge manoeuvres. There is no thruster aligned to produce a thrust in the Y axis direction(lateral). Kambara is therefore underactuated and not able to perform direct sway (lateral) motion.



**Figure 1.2: Kambara**

The upper enclosure houses a computer system, analog-to-digital converters, communications equipment and a sensor suite. The sensor suite consists of a triaxial accelerometer, digital compass, two digital inclinometers, rate-gyroscope, depth sensor and temperature sensor. The lower enclosure contains six lead-acid 12V batteries, power distribution and charging circuitry, and a leak sensor. Stereo cameras and a specialised sonar sensor can be mounted on the front frame of Kambara. All of the required processing power is present on-board Kambara.

Work previously completed on the control system for Kambara has incorporated a camera vision system. This control methodology utilising the visual targeting of objects within the pool environment was capable of stabilising the AUV and, tracking certain objects [16]. The Kambara project team has decided that the use of underwater vision is unreliable in most genuine conditions and has decided to rely on the inertial sensors for the fundamental control of the AUV. Additional information on obstacles within the environment will be provided by sonar technology.

A number of applications have been envisaged for Kambara, these include underwater mapping, observing marine life, explorations into the deep sea and under ice topped regions, and inspection and maintenance of underwater structures and cabling. These applications have use in the fields of marine biology,

---

marine geology, underwater inspection and assistance. The main aim of the Kambara project is to further underwater research particularly in the areas of underwater vehicle control and guidance.

### 1.3 Other AUV Research Projects

The area of AUV's is being researched by a number of Universities and research centers around the world. Currently a wide variety of AUV's are being developed with the aim to further utilise the expansive underwater environment. A few of the AUV's currently in use are described below, including aspects relating to the control methodologies implemented in the AUV's.

The "Oberon" is an AUV being developed at the Australian Centre for Field Robotics, located at the University of Sydney. This AUV was created from an identical design to Kambara, however its computation is performed on the surface through a tether. Recent work performed on the "Oberon" has been on control design from a terrain-aided navigation technique based on simultaneous localisation and map building. The low level control for this vehicle is made up of two independent PID controllers used to manage the vertical and horizontal motion of the AUV [15].

"ODIN" is an AUV that has been designed and built at the University of Hawaii. The AUV has a near-spherical shape with eight marine thrusters. Recent work performed on the AUV has been on adaptive control using quaternion based control laws. This control methodology attempts to adapt the control model during operation to accommodate the complex hydrodynamic effects [12].

The Department of Ocean Engineering at the Florida Institute of Technology is primarily researching an autonomous underwater data acquisition system that is capable of taking measurements of depths up to 6000 meters. This system will be based on an AUV that is capable of refuelling and exchanging data at an underwater station. This system is planning to utilise an experimental AUV that will use neural networks and fuzzy logic to control the vehicle [19].

The problem of hydrodynamic effects is the main area of current investigation and development into the control of AUV's.

### 1.4 Contributions

During my time at RSISE I have contributed the following work:

- Investigated all the components and operation of the inertia sensors onboard

---

Kambara. This has included ordering, wiring and mounting two inclinometers.

- Investigated and verified the system model for the Kambara system.
- Derived a feedback stability controller for Kambara based on the system model. This includes determination of the desired control parameters to achieve the required output response.
- Programmed the Accelus Amplifiers to receive a pulse width modulated(PWM) signal and to output the specified current to the thrusters. This has included optimising the control parameters to obtain a desirable step response.
- Developed a real time system design for the implementation of the code.
- Implemented the controller, incorporating the real time design, in the programming language Ada.
- Tested the implementation of the Ada code for both the control system and for real time functionality.
- Assisted with the wiring and the mounting of the new hardware system incorporating the amplifiers and sensors.
- Helped to debug the old code for Kambara which lacks certain functionality after an upgrade of the VxWorks operating system.

## 1.5 Project Objectives

The agreed objectives for the project were:

- To become familiar with the sensors onboard and to ensure that the Sensor Suite is capable of making the required inertial measurements.
- To develop a detailed system model for the Kambara System.
- To design a controller to stabilise Kambara.
- To implement a real time stabilising controller on Kambara using Ada under the VxWorks operating system.
- To test and demonstrate the operation of the controller.

---

## 1.6 Outline

The remainder of this thesis is specified as follows:

- **Chapter 2** – A system model for Kambara is detailed in this section. The attitude representation is described for both Euler Angles and Quaternions.
- **Chapter 3** – The sensor suite containing the Inertial Sensors of Kambara is investigated. The use and the physical specifications of each inertial sensor is detailed.
- **Chapter 4** – A vehicle controller is derived and presented. The controller aims to stabilise Kambara in the six degrees of freedom underwater environment.
- **Chapter 5** – The thruster controller utilising the amplifiers is defined.
- **Chapter 6** – The real time requirements for Kambara are specified, and the real time design is presented.
- **Chapter 7** – Results from the implementation of the controller and the real time system are presented and discussed. Possible avenues for further testing are also examined.
- **Chapter 8** – Conclusions and suggestions for future work for the inertial control of the Kambara system are presented.

Supporting technical material for the forgoing Chapters is presented as follows:

- **Appendix A** MotionPak Data Sheet
- **Appendix B** Thruster Output
- **Appendix C** Depth Sensor Testing
- **Appendix D** Ada Controller Code
- **Appendix E** Inclinator Testing Code
- **Appendix F** CD-ROM directory Structure

---

# System Model

---

In this chapter a mathematical model is detailed to describe the dynamic motion of the rigid body Kambara vehicle. This model will be used to develop the stability controller described in Chapter 4. The model is based on work previously described by Fossen [10] and by Silpa-Anan [16]. Given the complication of fluid dynamics in the six degrees of freedom problem posed by the Kambara vehicle, a number of nonlinear effects are present in the system. The cause of these effects include hydrodynamic drag, damping, and lift forces, Coriolis and centripetal forces. Gravity and buoyancy forces, although compensatable, are also described as factors occurring in the inertial control of Kambara.

An accurate model of the system is required to ensure that the Kambara system can be accurately and reliably controlled. A modification of equations derived by Silpa-Anan [16] is presented with a view to the creation of an inertial control system for Kambara.

## 2.1 Reference Frame

A two coordinate system is used to describe the motion and the location of the AUV. A world coordinate frame  $\{W\}$  is used to describe the position of the AUV in reference to global coordinates. A local reference frame representing Kambara  $\{K\}$  is fixed to the center of the rigid body. This frame is used to reference the inertial sensor readings onboard Kambara to the world coordinate frame. The reference frames are shown in Figure 2.1.

The reference frame for Kambara  $\{K\}$  utilises the X axis to represent the forward direction of the submersible. The Y axis is perpendicular to this axis with the positive direction towards the right of the submersible. The positive direction for the Z axis is downwards.



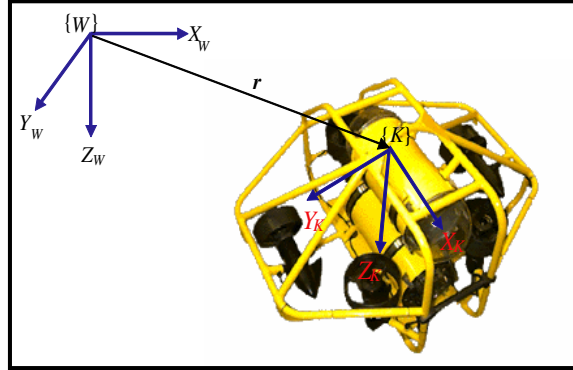


Figure 2.1: World and local Reference Frames

## 2.2 Attitude Representation

A number of different methods exist for the mathematical representation of the attitude (or orientation) for the AUV. Two well known and used forms are X-Y-Z Euler Angles and Quaternions. These methods are described below.

### 2.2.1 Euler Angles

The orientation of frame  $\{W\}$  in frame  $\{K\}$  is commonly specified in terms of three X-Y-Z Euler Angles: roll ( $\phi$ ), pitch ( $\theta$ ) and yaw ( $\psi$ ). These angles can be measured using inertial sensors onboard Kambara as described in Chapter 3. Figure 2.2 displays the Euler Angles.

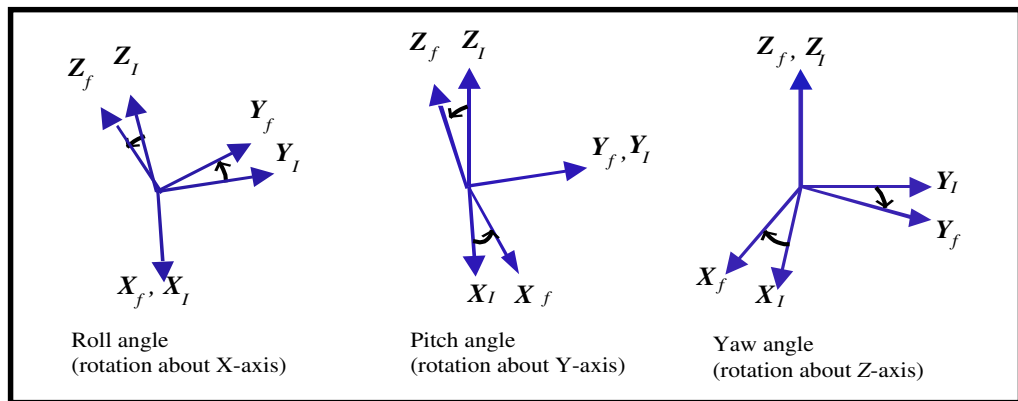


Figure 2.2: Euler Angles

It is necessary to be able to convert parameters stated in terms of the Kambara reference frame  $\{K\}$  into the world coordinate frame  $\{W\}$  parameters. In particular, one needs to convert vehicle velocity  $\mathbf{v}'$  and vehicle angular velocity  $\mathbf{w}'$  both expressed in terms of frame  $\{K\}$ , into world velocity and Euler Angle parameters. This can be performed by using the transformations  $\mathbf{v} = R(\phi, \theta, \psi)\mathbf{v}'$  and  $\mathbf{w} = T(\phi, \theta, \psi)\mathbf{w}'$ . These transformation matrices have been derived by Craig [7] and are shown in Equations 2.1 & 2.2

$$R(\phi, \theta, \psi) = \begin{pmatrix} \cos \theta \cos \psi & \sin \theta \sin \phi \cos \psi - \cos \phi \sin \psi & \sin \theta \cos \phi \cos \psi + \sin \phi \sin \psi \\ \cos \theta \sin \psi & \sin \theta \sin \phi \sin \psi + \cos \phi \cos \psi & \sin \theta \cos \phi \sin \psi - \sin \phi \cos \psi \\ -\sin \theta & \cos \theta \sin \phi & \cos \theta \cos \phi \end{pmatrix} \quad (2.1)$$

$$T(\phi, \theta, \psi) = \frac{1}{\cos \theta} \begin{pmatrix} \cos \theta & \sin \theta \sin \phi & \sin \theta \cos \phi \\ 0 & \cos \theta \cos \phi & -\cos \theta \sin \phi \\ 0 & \sin \phi & \cos \phi \end{pmatrix} \quad (2.2)$$

Euler Angle representations such as the one above have several disadvantages. All Euler Angle representations contain singularities, where the transformation will be undefined for some angles. In the X-Y-Z Euler angular transformation matrix (Equation 2.2), a singularity occurs at  $\theta = \pm \frac{\pi}{2}$ . As  $\theta$  approaches  $\pm \frac{\pi}{2}$ ,  $\tan \theta$  approach infinity and the transformation matrix is undefined. This results in a loss of numerical accuracy for Euler Angles close to  $\theta = \pm \frac{\pi}{2}$ .

An additional limitation of Euler Angles is with respect to computation time. The transformations require the calculation of six trigonometric functions, which are computationally expensive.

### 2.2.2 Quaternions

An alternate representation of the attitude that avoids the above problems can be obtained by use of quaternions. A quaternion representation uses a fourth parameter to eliminate the singularity problems of Euler Angles. It also eliminates the problem of evaluating trigonometric functions and provides for more stability. Once the first quaternion is calculated, quaternions need only be updated with a sequence of multiplications and additions.

Consider a quaternion  $\mathbf{q}$  shown in Equation 2.3, where  $\eta$  is a scalar value and  $\epsilon$  is a 3 element vector.

Define a unit vector  $\mathbf{k} = \begin{bmatrix} k_x & k_y & k_z \end{bmatrix}^T$  and an angle  $\theta$  such that rotation of the world coordinate frame  $\{W\}$  about  $\mathbf{k}$  by the angle  $\theta$  results in frame  $\{K\}$ .

$$\mathbf{q} = \begin{pmatrix} \eta \\ \epsilon \end{pmatrix} \quad (2.3)$$

where

$$\epsilon = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{pmatrix} \quad (2.4)$$

where

$$\eta = \cos \frac{\theta}{2} \quad (2.5)$$

$$\epsilon_1 = k_x \sin \frac{\theta}{2} \quad (2.6)$$

$$\epsilon_2 = k_y \theta \sin \frac{\theta}{2} \quad (2.7)$$

$$\epsilon_3 = k_z \sin \frac{\theta}{2} \quad (2.8)$$

Then, for example, a quaternion representing an orientation with frames  $\{W\}$  and  $\{K\}$  overlapping is  $\mathbf{q} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T$ .

The transformation equations involving quaternions are given by  $\dot{\mathbf{r}} = \mathbf{R}(\mathbf{q})\mathbf{v}'$  and  $\dot{\mathbf{q}} = \frac{1}{2}\mathbf{U}(\mathbf{q})\mathbf{w}'$ . From Fjellstad [9], the transformation matrices are given by Equations 2.10 and 2.11

$$\mathbf{R}(\mathbf{q}) = I_{3 \times 3} + 2 * \eta * \mathbf{S}(\epsilon) + 2[\mathbf{S}(\epsilon)]^2 \quad (2.9)$$

$$= \begin{pmatrix} 1 - 2\epsilon_2^2 - 2\epsilon_3^2 & 2(\epsilon_1\epsilon_2 - \eta\epsilon_3) & 2(\epsilon_1\epsilon_3 - \eta\epsilon_2) \\ 2(\epsilon_1\epsilon_2 + \eta\epsilon_3) & 1 - 2\epsilon_1^2 - 2\epsilon_3^2 & 2(\epsilon_2\epsilon_3 - \eta\epsilon_1) \\ 2(\epsilon_1\epsilon_3 - \eta\epsilon_2) & 2(\epsilon_2\epsilon_3 - \eta\epsilon_1) & 1 - 2\epsilon_1^2 - 2\epsilon_2^2 \end{pmatrix} \quad (2.10)$$

$$\mathbf{U}(\mathbf{q}) = \begin{pmatrix} -\epsilon^T \\ \eta I_{3 \times 3} + \mathbf{S}(\epsilon) \end{pmatrix} = \begin{pmatrix} -\epsilon_1 & -\epsilon_2 & -\epsilon_3 \\ \eta & -\epsilon_3 & \epsilon_2 \\ \epsilon_3 & \eta & -\epsilon_1 \\ -\epsilon_2 & \epsilon_1 & \eta \end{pmatrix} \quad (2.11)$$

Where the skew symmetric matrix operator  $\mathbf{S}$  is defined as

$$\mathbf{S}(\mathbf{a}) = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix} \quad (2.12)$$

The significance of such a matrix operator is that a vector cross product is

---

reduced down to matrix multiplication. Now as  $\mathbf{R}(\mathbf{q})$  and  $\mathbf{U}(\mathbf{q})$  have full rank, the equations contain no singularities. In addition the trigonometric functions have been eliminated from these transformation matrices. The quaternions allow for more stability as well as being a saving in computational efficiency.

### 2.2.3 Transformation from Euler Angles to Quaternions

The Kambara sensor suite directly measures the Euler Angles using the digital compass and the two inclinometers. A transformation is required to convert the Euler angles to Quaternions. The trigonometric functions must be determined to perform this conversion.

The transformation is shown in Equations 2.13 to 2.16. These equations were derived by Baker [2]

$$\eta = \cos \frac{\phi}{2} \cos \frac{\theta}{2} \cos \frac{\psi}{2} + \sin \frac{\phi}{2} \sin \frac{\theta}{2} \sin \frac{\psi}{2} \quad (2.13)$$

$$\epsilon_1 = \cos \frac{\phi}{2} \cos \frac{\theta}{2} \sin \frac{\psi}{2} - \sin \frac{\phi}{2} \sin \frac{\theta}{2} \cos \frac{\psi}{2} \quad (2.14)$$

$$\epsilon_2 = \cos \frac{\phi}{2} \sin \frac{\theta}{2} \cos \frac{\psi}{2} + \sin \frac{\phi}{2} \cos \frac{\theta}{2} \sin \frac{\psi}{2} \quad (2.15)$$

$$\epsilon_3 = \cos \frac{\phi}{2} \cos \frac{\theta}{2} \sin \frac{\psi}{2} - \sin \frac{\phi}{2} \sin \frac{\theta}{2} \cos \frac{\psi}{2} \quad (2.16)$$

## 2.3 Vector State Representation

This section describes the state space representation for the AUV.

### 2.3.1 Position State Vector

The position state vector  $\mathbf{P_K}$  is the current position of the submersible in the world coordinate frame. The position is described in terms of X-Y-Z coordinates and the orientation in Quaternions.

$$\mathbf{P_K} = \begin{pmatrix} \mathbf{x} \\ \mathbf{q} \end{pmatrix} \quad (2.17)$$

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (2.18)$$

$$\mathbf{q} = \begin{pmatrix} \eta \\ \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{pmatrix} \quad (2.19)$$

The position vector is specified by  $\mathbf{x}$  in meters and the orientation vector is specified by  $\mathbf{q}$  in Quaternions.

### 2.3.2 Velocity State Vector

The velocity state vector  $\mathbf{V_K}$  is the current velocity of the submersible in reference to the world coordinate frame.

$$\mathbf{V_K} = \begin{pmatrix} \mathbf{v} \\ \mathbf{w} \end{pmatrix} \quad (2.20)$$

$$\mathbf{v} = \begin{pmatrix} u \\ v \\ w \end{pmatrix} \quad (2.21)$$

$$\mathbf{w} = \begin{pmatrix} p \\ q \\ r \end{pmatrix} \quad (2.22)$$

The velocity vector is specified by  $\mathbf{v}$  in meters per second and the angular velocity vector is specified by  $\mathbf{w}$  in degrees per second.

### 2.3.3 Thrust State Vector

The thrust state vector  $\mathbf{T}$  is the current thrust of the submersible in terms of force and torque in reference to the world coordinate frame.

$$\mathbf{T} = \begin{pmatrix} \mathbf{F} \\ \mathbf{L} \end{pmatrix} \quad (2.23)$$

$$\mathbf{F} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad (2.24)$$

$$\mathbf{L} = \begin{pmatrix} K \\ M \\ N \end{pmatrix} \quad (2.25)$$

The force vector is specified by  $\mathbf{F}$  measured in Newtons and the torque vector is specified by  $\mathbf{L}$  measured in degrees per second<sup>2</sup>.

## 2.4 Vehicle State Representation

A detailed vehicle state representation for Kambara has been derived by Silpa-Anan [16], based on the general model proposed by Fossen [10]. The most significant of these equations, Equation 2.26, describes the hydrodynamic forces and torques created by the motion of Kambara in underwater operation, as follows:

$$\mathbf{T}_{RB} = \mathbf{T} - \mathbf{M}_A \dot{\mathbf{V}} - \mathbf{C}_A(\mathbf{V})\mathbf{V} - \mathbf{D}(\mathbf{V})\mathbf{V} - \mathbf{G} \quad (2.26)$$

Where:

$\mathbf{T}_{RB}$  is the thrust on the rigid body.

$\mathbf{T}$  is the force and torque from the thrusters.

$\mathbf{M}_A$  is the hydrodynamic added mass to the inertia matrix.

$\mathbf{C}_A$  is the Coriolis effect matrix.

$\mathbf{D}(\mathbf{V})$  is the hydrodynamic damping and lift.

$\mathbf{G}$  is the gravitational and buoyancy force and torque vector.

$\mathbf{V}$  is the velocity state vector.

$\dot{\mathbf{V}} = \frac{d\mathbf{V}}{dt}$ .

### 2.4.1 Thruster Forces and Torques

The relationship between the force and torque exerted on Kambara by the thrusters is given by Equation 2.27.

$$\mathbf{T} = \mathbf{L}\mathbf{U} \quad (2.27)$$

where  $\mathbf{U}$  is a vector of thrusts given by

$$\mathbf{U} = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix} \quad (2.28)$$

In the above equation:

$T_1$  is the thrust from the left horizontal thruster (LH),

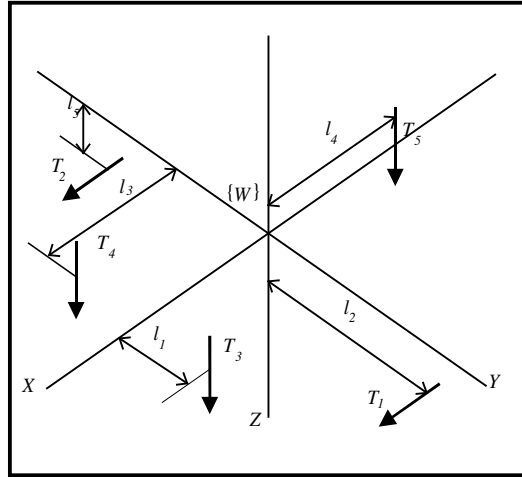
$T_2$  is the thrust from the right horizontal thruster (RH),

$T_3$  is the thrust from the front left vertical thruster (FLV),

$T_4$  is the thrust from the front right vertical thruster (FRV),

$T_5$  is the thrust from the rear vertical thruster (RV),

and are diagrammatically represented in Figure 2.3



**Figure 2.3: Thruster Layout**

and where  $\mathbf{L}$  is the thruster mapping matrix as defined in Section 2.4.8.

### 2.4.2 Mass and Inertia Matrix

The mass and inertia matrix is made up of the rigid mass and inertia of Kambara vehicle. These parameters, as described by Silpa-Anan [16], were obtained by

solid modelling of Kambara.

$$M_{RB} = \begin{pmatrix} 117 & 0 & 0 & 0 & 0 & 0 \\ 0 & 117 & 0 & 0 & 0 & 0 \\ 0 & 0 & 117 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10.7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 11.8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 13.4 \end{pmatrix} \quad (2.29)$$

### 2.4.3 Hydrodynamic Added Mass Matrix

This matrix represents the added mass to Kambara from the dynamic effects. As the vehicle moves, a finite amount of water is connected to the vehicle, adding additional mass. These effects were determined by Silpa-Anan [16], using experimental methods, as follows:

$$M_A \cong \text{diag} \left\{ 58.4 \quad 23.8 \quad 23.8 \quad 3.38 \quad 1.18 \quad 2.67 \right\} \quad (2.30)$$

### 2.4.4 Hydrodynamic Added Mass Coriolis-Like Matrix

Additional mass is added to the system by pressure induced forces and moments due to the forced motion of the rigid body vehicle. This effect is dependant on the added mass  $M_A$  and the vehicle state velocity. These effects were determined by Silpa-Anan [16] and may be expressed as follows:

$$C_A(\mathbf{v}) = \begin{pmatrix} 0 & 0 & 0 & 0 & 23.8w & -23.8v \\ 0 & 0 & 0 & -23.8w & 0 & 58.4u \\ 0 & 0 & 0 & 23.8v & -58.4u & 0 \\ 0 & 23.8w & -23.8v & 0 & 2.67r & -1.18q \\ -23.8w & 0 & 58.4u & -2.67r & 0 & 3.38p \\ 23.8v & -58.4u & 0 & 1.18q & -3.38p & 0 \end{pmatrix} \quad (2.31)$$

### 2.4.5 Hydrodynamic Damping Matrix

All the remaining hydrodynamic forces and moments are represented by this matrix.  $D(\mathbf{v})$  is typically a complex function of  $\mathbf{v}$ . In this case the matrix is a linear approximation of a quadratic lift plus drag and has been experimentally



derived by Silpa-Anan [16], as follows:

$$D(\mathbf{v}) = \begin{pmatrix} 120 + 90|u| & 0 & 0 & 0 & 0 & 0 \\ 0 & 90 + 90|v| & 0 & 0 & 0 & 0 \\ 0 & 0 & 150 + 120|w| & 0 & 0 & 0 \\ 0 & 0 & 0 & 15 + 10|p| & 0 & 0 \\ 0 & 0 & 0 & 0 & 15 + 12|q| & 0 \\ 0 & 0 & 0 & 0 & 0 & 18 + 15|r| \end{pmatrix} \quad (2.32)$$

#### 2.4.6 Gravitational and Buoyancy Forces Vector

This vector represents the overall effect that gravity and buoyancy have on the AUV. Before operation, the AUV needs to be calibrated by adding weights and floats to the open frame of the vehicle. Provided this calibration is undertaken, it is possible to disregard this force vector term in Equation 2.26

#### 2.4.7 Thrust Mapping Matrix

For purposes of mapping the AUV position, the force and torque vector needs to be adjusted so that it is in reference to the orientation of the individual thrusters and not in the world coordinate reference.

$L$  is a thrust mapping matrix which maps the producible force from the five thrusters to the force and torque required in the world coordinate frame from the thrusters. Its values, as experimentally derived by Silpa-Anan [16] are:

$$L = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & -0.28 & 0.28 & 0 \\ -0.05 & -0.05 & -0.32 & -0.32 & 0.43 \\ 0.47 & -0.47 & 0 & 0 & 0 \end{pmatrix} \quad (2.33)$$

Equation 2.27 can be rewritten to target the producible thrust  $U$ , as follows

$$U = L^\dagger T \quad (2.34)$$

---

Where

$$L^\dagger = (L^\top L)^{-1} L^\top \quad (2.35)$$

$$= \begin{pmatrix} 0.5 & 0 & 0 & 0 & 0 & 1.0638 \\ 0.5 & 0 & 0 & 0 & 0 & -1.0638 \\ -0.0333 & 0 & -0.2867 & -1.7857 & -0.6667 & 0 \\ -0.0333 & 0 & -0.2867 & 1.7857 & -0.6667 & 0 \\ 0.0667 & 0 & -0.4267 & 0 & 1.3333 & 0 \end{pmatrix} \quad (2.36)$$

## 2.5 Summary

This chapter describes the reference frames used to locate Kambara in the world coordinate frame. The attitude is described in both Euler Angles and Quaternions. The system model for Kambara is fully described including simplifications to the equations previously derived by Silpa-Anan [16] . A number of hydrodynamic effects are included in the model.

---

## Sensor Suite

---

The inertial control of Kambara is dependant on the correct operation of the inertial sensors onboard Kambara. The inertial sensors making up the Sensor Suite are detailed in Table 3.1.

Sensor Name	Manufacturer	Model No.	Output type
<b>Compass Module</b>	Precision Navigation	TCM2-50	Digital or Analog
<b>Inclinometer</b>	US Digital	A2I	Digital
<b>Pressure Sensor</b>	SensorTechnics	PTE2005G1A	Analog
<b>MotionPak</b>	Systron Donner	MP-GDDDDQBBB-100	Analog

**Table 3.1: Kambara Sensor Suite**

All of the sensors are located in the top enclosure of the hull next to the CPU. A picture of the sensor suite is shown in Figure 3.1. The pressure sensor is not visible as it is mounted into the outer wall of the hull.



**Figure 3.1: Kambara Sensor Suite**

### 3.1 Digital Compass - TCM2-50

The TCM2 compass module contains a biaxial electrolytic inclinometer and a triaxial magnetometer system. The purpose of this sensor is to measure the attitude in Euler Angles. The biaxial inclinometer has no mechanical moving parts and it uses a fluid filled tilt sensor, which is an angle sensing device that uses gravity as a reference to measure the orientation of the compass. The heading or yaw orientation information is determined from the magnetometer system. The specifications for the digital compass are shown in Table 3.2.

	Heading	Tilt	Magnetometer
<b>Full scale range</b>	0...360°	±50°	±80μT
<b>Accuracy - Tilted</b>	±1.5°	±0.4°	±0.2μT
<b>Resolution</b>	0.1°	0.3°	0.01μT
<b>Accuracy - Tilted</b>	±0.3°	±0.3°	±0.2μT
<b>Frequency</b>	16Hz		
<b>Operating range</b>	-20°C ... 70°C		

**Table 3.2: Compass Specifications**

There are three major limitations to this module, namely :-

- Diminished performance under acceleration;
- Limited tilt range of 50 degrees; and
- Sampling Speed.

The fluid filled tilt sensor is the fundamental cause of these limitations. It is inaccurate in situations where there are sudden changes in heading, as the result of the fluid in the inclinometer becoming turbulent (or sloshing). The settling time for the sloshing fluid is 300ms. Acceleration exerted on the fluid will also cause it to tilt due to inertial effects, causing a decrease in accuracy. A relationship between acceleration and tilt error is provided by the manufacturer in the TCM2 user's manual. In order to help minimise these error problems, two 360 ° inclinometers were installed in the AUV to perform this function.

The triaxial magnetometer system is utilised in the inertial sensor suite for the control of Kambara. There is a potential error from magnetic field distortion caused by the close operation of the triaxial magnetometer system, to that of the CPU. This device must be calibrated to adjust for these effects. In addition, the thrusters onboard Kambara create a varying magnetic field. These varying effects are some distance away from the sensor and are considered insignificant.

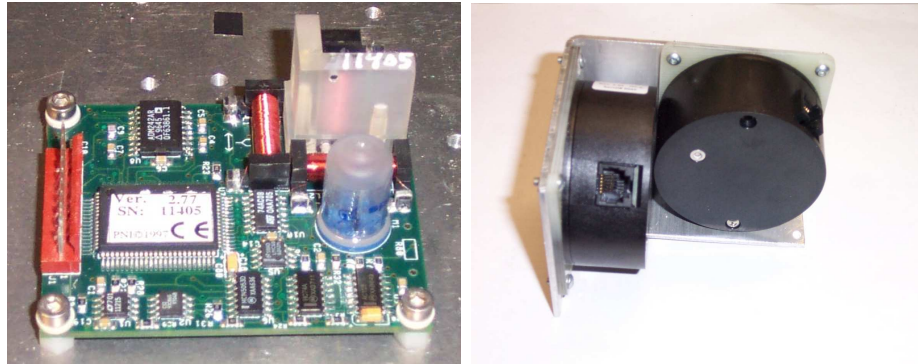


Figure 3.2: Compass Sensor(left) & Inclinometers(right)

### 3.2 360° Absolute Inclinometers

The inclinometers are a new addition to the sensor suite of Kambara. The inclinometers, made by US Digital, measure the full 360° degree range for both the pitch and yaw angles. The sensor measures the absolute angle value referencing gravity as opposed to the less accurate incremental sensors. Each sensor measures the rotational angle on a single axis in reference to gravity. The inclinometers are used in addition to the magnetometer to measure the attitude of Kambara. The device communicates by digital signals through the serial port. The specifications for the inclinometers are shown in Table 3.3.

<b>Operating Range</b>	0.0 ... 359.9°
<b>Position Error</b>	$\pm 0.3^\circ$ typical, $\pm 0.3^\circ$ max
<b>Settling time</b>	250 ms
<b>Position Update Rate</b>	8 ms
<b>Operating Temperature</b>	-25°C ... 70°C

Table 3.3: Inclinometer Specifications

The inclinometers have been selected to minimise the errors created in measuring the pitch and yaw angles. They have been wired and mounted into the sensor suite. Testing has been performed on the sensors to ensure correct operation. In order to perform this testing, specific commands were sent to the sensors using specially created Ada code as shown in Appendix E. The inclinometers were tilted and the angle of rotation was compared to the sensor readings. The sensors produced output to the accuracy specified in the manufacturer's data sheets.

### 3.3 MotionPak

The Systron Donner MotionPak is a 6 DOF inertial sensing system. It contains three orthogonally mounted linear servo accelerometers, three micromachined quartz angular rate gyroscopes and a temperature sensor. The Motion Pak produces output in the vehicles reference frame  $\{K\}$ . These readings must later be translated to the world coordinate frame  $\{W\}$ . The specifications for the MotionPak are shown in Table 3.4.

	Gyroscope	Accelerometer
<b>Full scale range</b>	$\pm 2g$	$\pm 2g$
<b>Output voltage swing</b>	$\pm 2.5V$	$\pm 7.5V$
<b>Sensitivity</b> At 22 °C Temperature	$25mV/^{\circ}/s$ $< 0.03\%/^{\circ}C$	$3.75V/g$ $< 0.03\%/^{\circ}C$
<b>Bias</b> At 22 °C Drift 22 °C to TMAX	$0^{\circ}/s$ $< 3^{\circ}/s$	$0g$ $< 100\mu g/^{\circ}C$
<b>Bandwidth (-90 )</b>	$> 60Hz$	$> 300Hz$
<b>Operating range</b>	$-40^{\circ}C \dots 80^{\circ}C$	

Table 3.4: MotionPak Specifications

The MotionPak requires calibration by a scale factor and a bias for each value it generates. The manufacturer has provided a data sheet for this particular sensor as shown in Appendix A. Testing of this sensor by Cvetanovski [8] produced significantly improved scale factors for the acceleration values. A summary of the scale factors and biases are shown in Table 3.5

Axis	Scale Factor	Bias
<b>Linear X*</b>	$2.611203 \text{ } m/s^2/V$	$0.036094 \text{ } m/s^2$
<b>Linear Y*</b>	$2.660585 \text{ } m/s^2/V$	$0.056003 \text{ } m/s^2$
<b>Linear Z*</b>	$2.670453 \text{ } m/s^2/V$	$-0.033754 \text{ } m/s^2$
<b>Angular X**</b>	$40.31608 \text{ } ^{\circ}/s/V$	$0.10 \text{ } ^{\circ}/s$
<b>Angular Y**</b>	$40.12519 \text{ } ^{\circ}/s/V$	$0.01 \text{ } ^{\circ}/s$
<b>Angular Z**</b>	$40.09241 \text{ } ^{\circ}/s/V$	$0.07 \text{ } ^{\circ}/s$

\* From p21, Cvetanovski [8]

\*\* Derived from Appendix A

Table 3.5: MotionPak Calibration Values



Figure 3.3: MotionPak(left) &amp; Pressure Sensor(right)

### 3.4 Pressure Sensor

The pressure sensor is used to measure the depth in the Z coordinate of Kambara in the world coordinate frame. The sensor is made by SensorTechnics and is encased in stainless steel. The specifications for the pressure sensor are shown in Table 3.6.

<b>Full scale range</b>	0 ... 3.5m
<b>Output Range</b>	1 ... 6V
<b>Bias</b>	1V
<b>Scale Factor(25°C)</b>	0.7V/m
<b>Thermal Effects</b>	Typ $\pm 0.6\%$ , Max 2.5% FSO
<b>Non-linearity</b>	Typ $\pm 0.2\%$ , Max 0.5% FSO
<b>Repeatability</b>	$\pm 0.1\%$ FSO
<b>Output Noise</b>	0.04% FSO
<b>Cutoff frequency</b>	1kHz
<b>Operating range</b>	0°C ... 70°C

FSO – Full Scale Output

Table 3.6: Pressure Sensor Specifications

The pressure sensor must be mounted into the side of the hull as it requires contact with the water to measure the pressure accurately. Testing performed on the sensor by Beswick [3] shows that the device is sensitive to changes of 5 mm in depth. The results are shown in Appendix C. Note that this sensor has been deliberately chosen to have a limited depth range so that greater accuracy

could be achieved in shallow water testing. For additional testing in the sea environment this sensor will need to be replaced by one with a greater depth range.

### 3.5 Sensor Communications

To utilise the sensors onboard Kambara the controller operating on the CPU must be able to reliably access the data from the sensors in real time. The data must be in a form that accurately represents the measured values, without excessive signal noise. As shown in Table 3.1, there are two types of outputs from the sensors, these being analog and digital.

To access the digital sensor outputs, Kambara has a serial port allowing access to two digital RS-292 data channels. The digital compass and inclinometers require these ports to transmit their measured values. These sensors must be initialised from the CPU with calibration information and with the required sensor data, which the CPU will be expecting. Once initialised, the compass unit will continue to automatically return readings of its current position through the serial connection. The inclinometers, however, need to be sent a request command from the CPU. Once the inclinometers receive this command the current position will be measured and returned through the serial connection.

For the two analog sensors, a more detailed process is required to obtain the data from the sensors. The process is as follows:

- Filter the analog output signal through a low pass filter;
- Convert the analog data using an analog to digital converter (ADC); and
- Adjust the value using the specified bias and scale factor values.

The use of a low pass filter to improve the signal produced by these devices was investigated by Bethlehem [4]. The implemented low pass filters remove the spikes in the output of the MotionPak rate gyroscope and remove the high frequency, white Gaussian noise that occurs in the system.

The analog to digital conversion is performed by the iPADIO card which is capable of converting 50 channels of data. To ensure that the final value obtained by the controller is the correct measured value, each reading must be adjusted by Equation 3.1, as follows:

$$\text{FinalValue} = \text{ScaleFactor} * \text{Voltage} + \text{Bias} \quad (3.1)$$

The communication between the different components of Kambara is represented by Figure 3.4



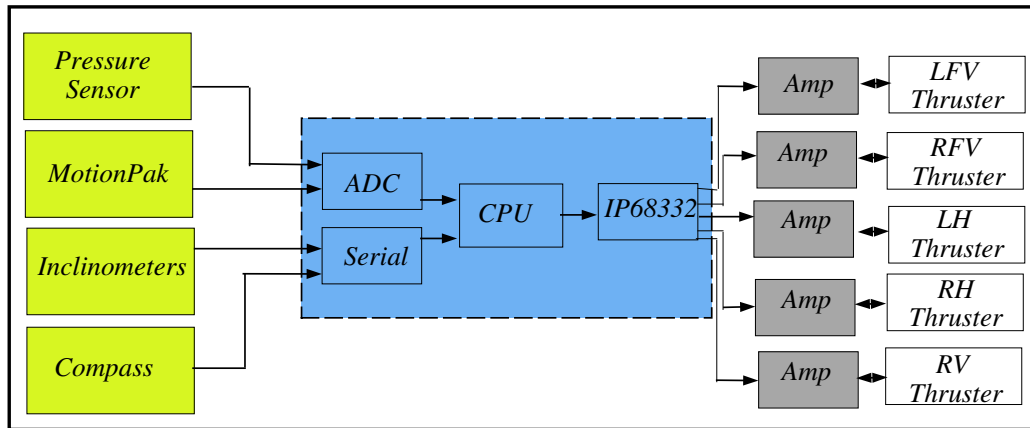


Figure 3.4: Communication of Sensor and Output Data

---

# Kambara Controller

---

The inertial control of the Kambara system is dependant on the correct operation of the inertial sensors and the correctness of the system model. The aim of the Kambara controller is to:

- Stabilise the Kambara vehicle in the six degrees of freedom environment;
- Track the AUV to a target position, denoted by a reference signal specified by the user; and to
- Execute control of the vehicle in the real time environment.

This chapter derives and presents the control system for Kambara.

## 4.1 Controller Derivation

Following the method outlined in Franklin [11], and specifically the Laplace Transform methodology, a mathematical description for the AUV is derived.

Given the System model detailed in Chapter 2, Equation 2.26 can be rewritten, targeting the force and torque vector of the thrusters, as follows:

$$\mathbf{T}_{RB} = \mathbf{T} - \mathbf{M}_A \dot{\mathbf{V}} - \mathbf{C}_A(\mathbf{V})\mathbf{V} - \mathbf{D}(\mathbf{V})\mathbf{V} - \mathbf{G} \quad (4.1)$$

$$\mathbf{T} = \mathbf{T}_{RB} + \mathbf{M}_A \dot{\mathbf{V}} + \mathbf{C}_A(\mathbf{V})\mathbf{V} + \mathbf{D}(\mathbf{V})\mathbf{V} + \mathbf{G} \quad (4.2)$$

Using the position vector described in Chapter 2, namely:

$$\mathbf{P} = \begin{pmatrix} \mathbf{x} \\ \mathbf{q} \end{pmatrix} \quad (4.3)$$

$$\dot{\mathbf{P}} = \mathbf{V} = \begin{pmatrix} \mathbf{v} \\ \mathbf{w} \end{pmatrix} = \frac{d\mathbf{P}}{dt} \quad (4.4)$$

$$\ddot{\mathbf{P}} = \dot{\mathbf{V}} = \frac{d\mathbf{V}}{dt} \quad (4.5)$$

and given that the force on the rigid body is described by the following

$$\mathbf{T}_{RB} = \mathbf{M}\ddot{\mathbf{P}} = \begin{pmatrix} M_x & 0 & 0 & 0 & 0 & 0 \\ 0 & M_y & 0 & 0 & 0 & 0 \\ 0 & 0 & M_z & 0 & 0 & 0 \\ 0 & 0 & 0 & J_\psi & 0 & 0 \\ 0 & 0 & 0 & 0 & J_\theta & 0 \\ 0 & 0 & 0 & 0 & 0 & J_\phi \end{pmatrix} \ddot{\mathbf{P}} \quad (4.6)$$

Let

$$\mathbf{A} = \mathbf{M}_A + \mathbf{M} \quad (4.7)$$

And

$$\mathbf{B} = \mathbf{C}_A(\mathbf{V}) + \mathbf{D}(\mathbf{V}) \quad (4.8)$$

Substituting Equations 4.7 and 4.8 into Equation 4.2 yields

$$\mathbf{T} = \mathbf{A}\ddot{\mathbf{P}} + \mathbf{B}(\dot{\mathbf{P}})\dot{\mathbf{P}} + \mathbf{G} \quad (4.9)$$

Now an expression for the relationship of the forces managed by the controller of the AUV is presented, namely:

$$\mathbf{T} = -\mathbf{A}(\mathbf{k}_1\dot{\mathbf{P}} + \mathbf{k}_2\mathbf{P}) + \mathbf{B}(\dot{\mathbf{P}})\dot{\mathbf{P}} + \mathbf{G} \quad (4.10)$$

By substituting for  $\mathbf{T}$  in Equations 4.10 and 4.9, we obtain

$$-\mathbf{A}(\mathbf{k}_1\dot{\mathbf{P}} + \mathbf{k}_2\mathbf{P}) + \mathbf{B}(\dot{\mathbf{P}})\dot{\mathbf{P}} + \mathbf{G} = \mathbf{A}\ddot{\mathbf{P}} + \mathbf{B}(\dot{\mathbf{P}})\dot{\mathbf{P}} + \mathbf{G} \quad (4.11)$$

Leaving

$$-\mathbf{A}(\mathbf{k}_1\dot{\mathbf{P}} + \mathbf{k}_2\mathbf{P}) = \mathbf{A}\ddot{\mathbf{P}} \quad (4.12)$$

As  $\mathbf{A}$  is a positive diagonal matrix it is invertible, giving

$$\mathbf{0} = \ddot{\mathbf{P}} + \mathbf{k}_1\dot{\mathbf{P}} + \mathbf{k}_2\mathbf{P} \quad (4.13)$$

Taking the Laplace Transform, Equation 4.13 may be expressed as

$$\mathbf{0} = s^2\mathbf{P}(s) - s\mathbf{P}(0) - \dot{\mathbf{P}}(0) + \mathbf{k}_1s\mathbf{P}(s) - \mathbf{k}_1\mathbf{P}(0) + \mathbf{k}_2\mathbf{P}(s) \quad (4.14)$$

Therefore

$$\mathbf{P}(s) = \frac{\dot{\mathbf{P}}(0) + (s + \mathbf{k}_1)\mathbf{P}(0)}{s^2 + \mathbf{k}_1s + \mathbf{k}_2}, \quad (4.15)$$

where the parameters  $k_1$  and  $k_2$  are control parameters, that can be adjusted so that the output performs as required.

## 4.2 Reference Values

In order to make the controller track to a reference value for the position, velocity and acceleration, an alternative relationship is now derived for the main controller equation (Equation 4.10).

Let the reference  $\mathbf{r}$  be represented by

$$\mathbf{r} = \begin{pmatrix} r_x \\ r_y \\ r_z \\ r_\phi \\ r_\theta \\ r_\psi \end{pmatrix} \quad (4.16)$$

As we are designing a stability controller, set the values of  $\dot{\mathbf{r}}$  and  $\ddot{\mathbf{r}}$  to 0.

$$\dot{\mathbf{r}} = \ddot{\mathbf{r}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad (4.17)$$

Including the reference  $\mathbf{r}$  into the controller specified in Equation 4.10 yields

$$\mathbf{T} = -\mathbf{A}(\mathbf{k}_1\dot{\mathbf{P}} + \mathbf{k}_2(\mathbf{P} - \mathbf{r})) + \mathbf{B}(\dot{\mathbf{P}})\dot{\mathbf{P}} + \mathbf{G} \quad (4.18)$$

And substituting into Equation 4.9

$$-\mathbf{A}(\mathbf{k}_1\dot{\mathbf{P}} + \mathbf{k}_2(\mathbf{P} - \mathbf{r})) + \mathbf{B}(\dot{\mathbf{P}})\dot{\mathbf{P}} + \mathbf{G} = \mathbf{A}\ddot{\mathbf{P}} + \mathbf{B}(\dot{\mathbf{P}})\dot{\mathbf{P}} + \mathbf{G} \quad (4.19)$$

Leaving

$$-\mathbf{A}(\mathbf{k}_1\dot{\mathbf{P}} + \mathbf{k}_2(\mathbf{P} - \mathbf{r})) = \mathbf{A}\ddot{\mathbf{P}} \quad (4.20)$$

As  $\mathbf{A}$  is a positive diagonal matrix it is invertible

$$\mathbf{0} = \ddot{\mathbf{P}} + \mathbf{k}_1\dot{\mathbf{P}} + \mathbf{k}_2(\mathbf{P} - \mathbf{r}) \quad (4.21)$$

Let

$$\mathbf{e} = \mathbf{P} - \mathbf{r} \quad (4.22)$$

As  $\dot{\mathbf{r}}$  and  $\ddot{\mathbf{r}} = 0$ , the simplified equation becomes

$$\mathbf{0} = \ddot{\mathbf{e}} + \mathbf{k}_1\dot{\mathbf{e}} + \mathbf{k}_2\mathbf{e} \quad (4.23)$$

Now we can choose the control parameters  $k_1$  and  $k_2$  to shape the error function. The error function is the difference between the target reference value and the current value. The aim is to minimise the error function by reducing rise time, without causing overshoot. The control parameter design is investigated in Section 4.3.

The user of the control system can now specify the reference signal to be used as input. When the controller receives the reference signal it will attempt to track the reference signal. In the case of the position vector, the controller will attempt to reach both the position and the orientation specified in the reference signal. Currently the X and Y position values cannot be tracked as there is no way of accurately measuring these values using inertial sensors.

This controller can be adjusted to track the velocity in addition to the position of the vehicle. In this case the velocity can be obtained by integrating the acceleration values obtained from the MotionPak, while the angular velocities can be directly measured by the MotionPak. So a reference can be specified by the user and the error function for the tracking of the velocity state vector can be obtained.

The acceleration cannot be tracked using this method. There is no accurate way of determining the angular acceleration from the sensor readings. The value can be approximated by calculating the change of velocity over a given time period and establishing its average value.

Once Kambara is fully autonomous the reference value will be updated by a different system. This system would be a navigation system which controls the current position and the velocity of the AUV. Current work is being undertaken using Sonar technology to generate and record local area information. This information can be used to navigate in the underwater environment.

The compass, inclinometers and pressure sensors update the values of the current position. These values return into the control system providing feedback of how well the controller is tracking the reference values. These values must be gathered and ready to use when required by the controller.

The output of this model is in the form of thrust values for regulation of the thrusters by the amplifiers. The hardware IP68322 card is responsible for transmitting PWM values to the thrusters. A low level software interface exists between the CPU and this card. This interface is not currently working as further described in Chapter 7, a situation which has prevented dynamic testing of the AUV control system described in this chapter.

The control diagram illustrating the feedback loop for the Kambara control system is shown in Figure 4.1.

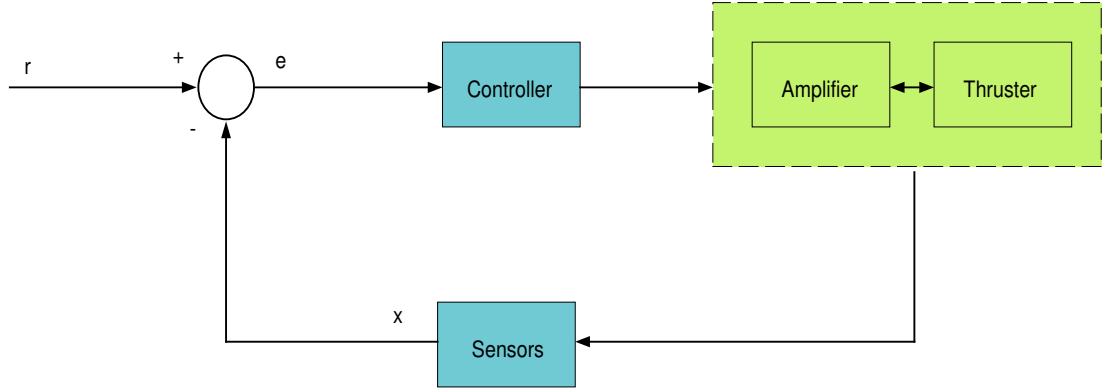
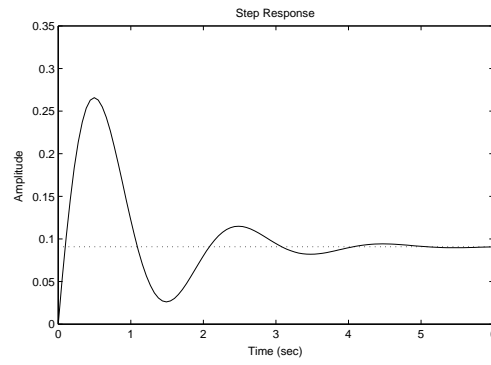
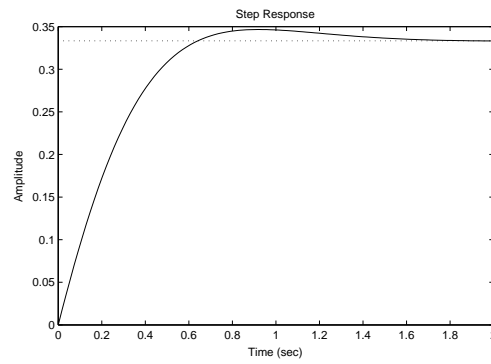
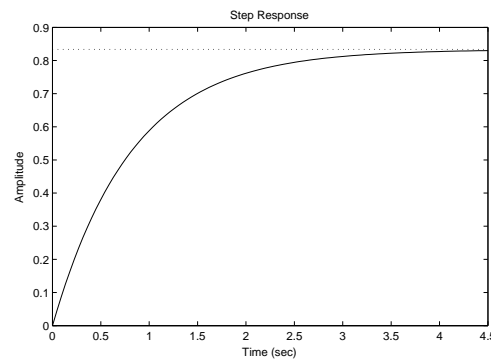
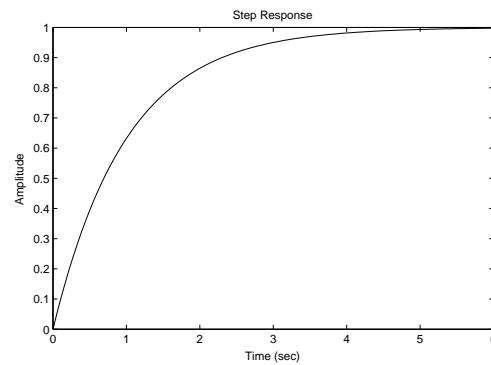


Figure 4.1: Feedback Control System for Kambara

### 4.3 Control Parameter Design

The control parameters  $k_1$  and  $k_2$  have been investigated to obtain optimal values for the output of the system. It is important that the system is able to track the reference value with a minimal steady state error. In addition overshoot is to be avoided, as the thrusters will have to change the direction of their output when the overshoot occurs. A fast rise time is not crucial in this analysis as the AUV is not required to quickly reach the target values.

Simulations have been performed using the computer program Matlab, incorporating the the final control design, represented by Equation 4.23. The results of the simulations were gathered and compared. A selection of the results are shown in Figures 4.2 to 4.5. The design parameters of  $k_1 = 10$  and  $k_2 = 0.1$  were chosen for the control parameters, as these parameters produced the best performing step response of the output (Figure 4.5). With these parameters, the step response has a small steady state error, no overshoot and an acceptable rise time.

Figure 4.2: System Step Response:  $k_2 = 10, k_1 = 1$ Figure 4.3: System Step Response:  $k_2 = 5, k_1 = 10$ Figure 4.4: System Step Response:  $k_2 = 1, k_1 = 2$ Figure 4.5: System Step Response:  $k_2 = 0.1, k_1 = 10$

---

# Thruster Controller

---

In addition to the controller for Kambara, which generates the required thrust commands, further control is required to ensure that the thrusters efficiently reach and accurately maintain the required thrust. This control function is represented by the green section in Figure 4.1.

## 5.1 Thrusters

The five thrusters are Minnkota Turbo Pro 324 units, which operate at 24V. A large continuous current of up to 10A is required by the thrusters to produce a thrust of approximately 60 N. To achieve this level of continuous current an amplifier is required. A data sheet showing the performance of this motor under different operating conditions is provided at Appendix B.

## 5.2 Accelus Amplifiers

The previous amplifiers that were used to output the current to the thrusters were originally made by students and mounted on the cylindrical frame of Kambara. Recently, a decision was made to replace the amplifiers with commercial models that could be mounted inside the hull of Kambara. The new amplifiers, are the Accelus model made by Copley Controls Corp and were chosen so that they could produce a high level of continuous current. Summary information detailing the performance characteristics of these amplifiers is shown in Table 5.1, and the unit is illustrated in Figure 5.1

These amplifiers have a number of built-in features for DC brush and brush-less motors. They need to be calibrated by programming with detailed motor information. The amplifier has non-volatile flash memory, which it uses to store the motor data. These values are used to maximise the efficiency of the motors by generating the highest motor torque over a wide speed range.



---

<b>Model</b>	Accelus ASP-090-36
<b>Continuous Current</b>	12A
<b>Peak Current</b>	36A
<b>Continuous Output Power</b>	1.0 kWatts

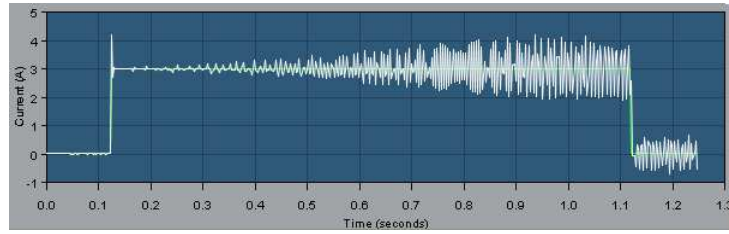
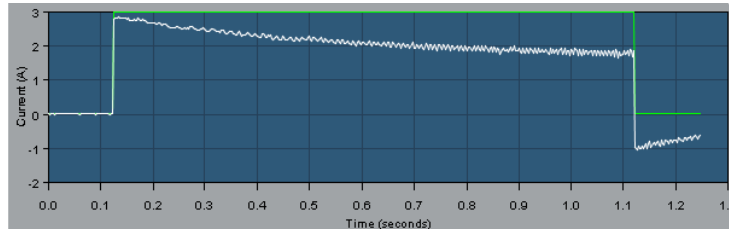
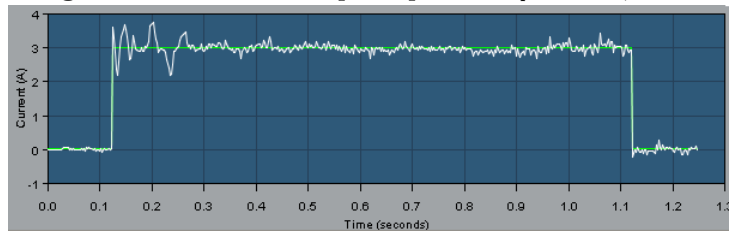
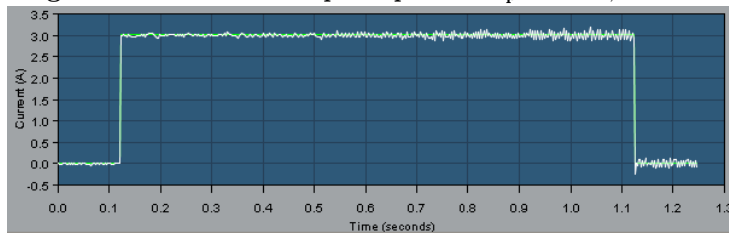
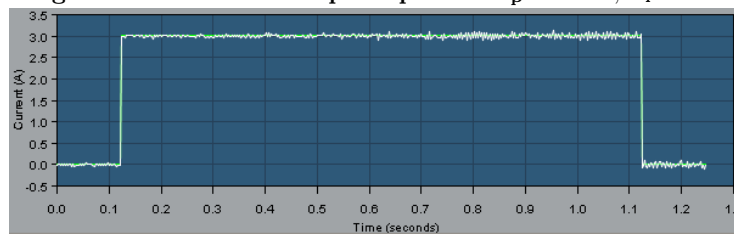
**Table 5.1:** Accelus Amplifier Summary Information**Figure 5.1:** Accelus Amplifier

The inputs to the amplifiers must also be specified. The amplifiers have been programmed to receive a 100% pulse width modulated(PWM) signal, a directional signal and an enable signal. The input signals are generated by the IP68332 card as specified in Section 4.2. The amplifiers generate a current for the motors of the thrusters. An error message is sent to the CPU, should any errors occur during operation.

### 5.3 Thruster Control System

These amplifiers have a built-in PID control system for regulation of the velocity, position or current of the motors. Feedback is required for these control loops, typically obtained using encoders on the thrusters. No encoders are present on the thrusters of the AUV, so only the current can be controlled using the back EMF current as feedback. As a result the amplifiers had to be programmed using the current feedback loop. Two control parameters had to be specified by the user to ensure that the controller performed as required. These control parameters are specified by  $C_p$  and  $C_i$ .

Testing has been performed to help determine the optimal values for these control parameters. A thruster was wired to an amplifier and the step response for a change in current was measured on a digital oscilloscope. The results of this testing were gathered and compared. A selection of the results are shown in Figures 5.2 to 5.6. The design parameters of  $C_p = 2000$  and  $C_i = 500$  were chosen for the control parameters, as these results produced the best performing step response (Figure 5.6).

Figure 5.2: Thruster Step Response:  $C_p = 100, C_i = 50$ Figure 5.3: Thruster Step Response:  $C_p = 1500, C_i = 0$ Figure 5.4: Thruster Step Response:  $C_p = 3000, C_i = 500$ Figure 5.5: Thruster Step Response:  $C_p = 1500, C_i = 200$ Figure 5.6: Thruster Step Response:  $C_p = 2000, C_i = 500$

---

# Real Time Analysis

---

Kambara is required to operate in an environment where its real time operational factors are crucial for reliable and accurate control results. The Kambara controller requires access to the most recent readings from the inertial sensors. It must also quickly make decisions on how to control the vehicle based on these values. Failure to meet these demands results in the processing of incorrect data by the controller and the transmission of incorrect values to the amplifiers, making the system unstable. To ensure that the system is able to operate in the real time environment the following requirements must be met:

- To retrieve the 10 different sensor values from the sensors arriving at different frequencies;
- To perform the calculations required in the control system on the most recent sensor data;
- To operate within a time period of 0.2s, to ensure that the controller is able to quickly respond to changes; and
- To efficiently output values to the amplifiers.

## 6.1 Operating System and Programming Language

To ensure that Kambara can operate effectively in the real time environment, the operating system VxWorks, running the programming language Ada, has been selected. Both the operating system and programming language offer well proven real time capabilities and support by providing predictability.

The development of the operating system, based on VxWorks is performed on its host machine, thereby making it ideal for Kambara. This is because the development is performed externally on more powerful machines and then downloaded when required for use. As an example, the U.S. National Aeronautics and

Space Administration's (NASA) "Mars Exploration Rovers" project utilises this operating system due to its reliability under extreme circumstances [14].

The Ada programming language provides not only features commonly found in other programming languages, but also provides additional support for controlling hardware to meet real time deadlines. Some current applications utilising the real time features in Ada include: commercial jets; air traffic control systems; NASA's Space Shuttle and Space Station Environment; automated manufacturing systems; continuous medical monitoring systems and real-time embedded control of copier and duplicator products [1]. These projects demonstrate the reliable, real time capabilities of Ada in a practical, applied sense.

## 6.2 Real Time System Design

To meet the real time design requirements, a real time implementation approach has been developed. The design attempts to ensure that Kambara will correctly execute in the real time environment without any timing implications. The design also utilises a number of real time features built into the Ada programming language. The real time features must be carefully monitored to ensure that they do not introduce unpredictable synchronisation effects to the system.

The design uses multiple tasks or threads running concurrently on the CPU. These tasks each place a demand on the CPU when they have work that needs computation. Four tasks have been implemented in the design where three tasks are responsible for gathering sensor readings and the fourth runs the control routine. These tasks all access one protected object called the protected sensor object. This arrangement is illustrated in Figure 6.1.

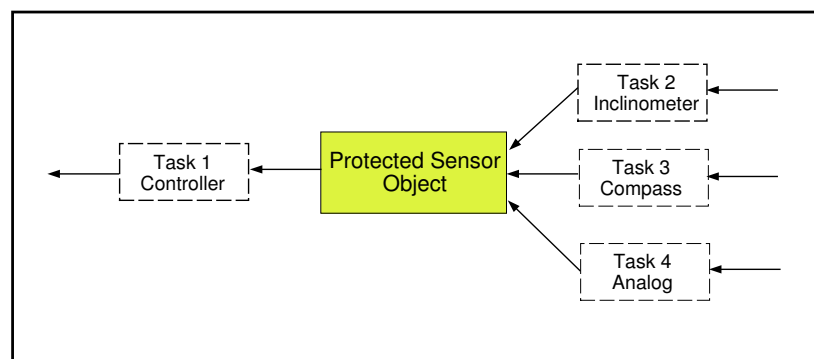


Figure 6.1: Real Time System Data Flow

The protected sensor object is the centre of the real time design. This object

---

is responsible for keeping track of the latest sensor readings. Using the protected feature of Ada ensures that only one task can access the object at any given time. If another task wants to access the object while it is currently being accessed by another task, then the task must wait until the object is free. This ensures that the data is always valid, by ensuring that multiple tasks cannot be updating values at the same time, while the controller task is trying to retrieve information on the current sensor readings.

Three of the tasks are responsible for gathering the sensor readings and updating the protected object. These tasks will get informed when an interrupt occurs in the system, notifying that the appropriate port is waiting to be read. Each task, upon receiving a relevant interrupt signal will read the value from the appropriate port. Any adjusting or filtering that needs to be applied to these signals will be performed by the task, and the task will then update the protected object.

The remaining task which performs all the calculations for the control system is the controller task. This task commences by reading a complete set of the sensor readings from the protected object. The task then performs a number of calculations on the sensor values and outputs appropriate values to the amplifiers. This task is delayed at the end of its execution to ensure that it runs to a specified time period. The time period is specified in the mathematical expressions and ensures that all of the sensor values are updated at least once before the values are recalculated. The synchronisation of these tasks is examined in Chapter 7.

---

# System Testing

---

This chapter examines the implementation of the code for the control system specified in Chapter 4 and the real time analysis aspects specified in Chapter 6, with a view to accurately control the Kambara system. The response of the controller from a full range of sensor inputs is initially examined. The real time aspects are then examined by investigating the occurrence of the major events in individual components of the system and, finally, the possibility of real time pool testing of the AUV is discussed.

## 7.1 Controller Response

To ensure that the system implementation is working as expected, a number of tests have been performed. These tests aim to ensure that, as the different sensor readings enter the system, the controller reacts by producing appropriate outputs to the thrusters. The final version of the code, used to generate these values, is listed in Appendix D. The code was generated using the texts [6] and [5].

For each sensor value a full range of the possible inputs to the system have been created. They have been introduced into the controller in a similar method to that of the real time sensor readings. The effect on the output to the five thrusters is measured and examined for each sensor value that is used in the Kambara controller. The thruster layout of Kambara has been previously described (See Figure 2.3).

### 7.1.1 Acceleration Testing

The acceleration values are updated by the MotionPak sensor. The reaction force of the thrusters should be increasing as the acceleration increases and should be in the opposing direction. As there are no thrusters in the Y direction, the motion in this direction is not controlled. A separate controller could be developed to attempt to oppose these forces. However these forces in past testing have not

been significant, as shown by Silpa-Anan [16]. The results of testing performed on the acceleration values are shown in Figure 7.1 and, at the end of this Chapter, in Figure 7.5.

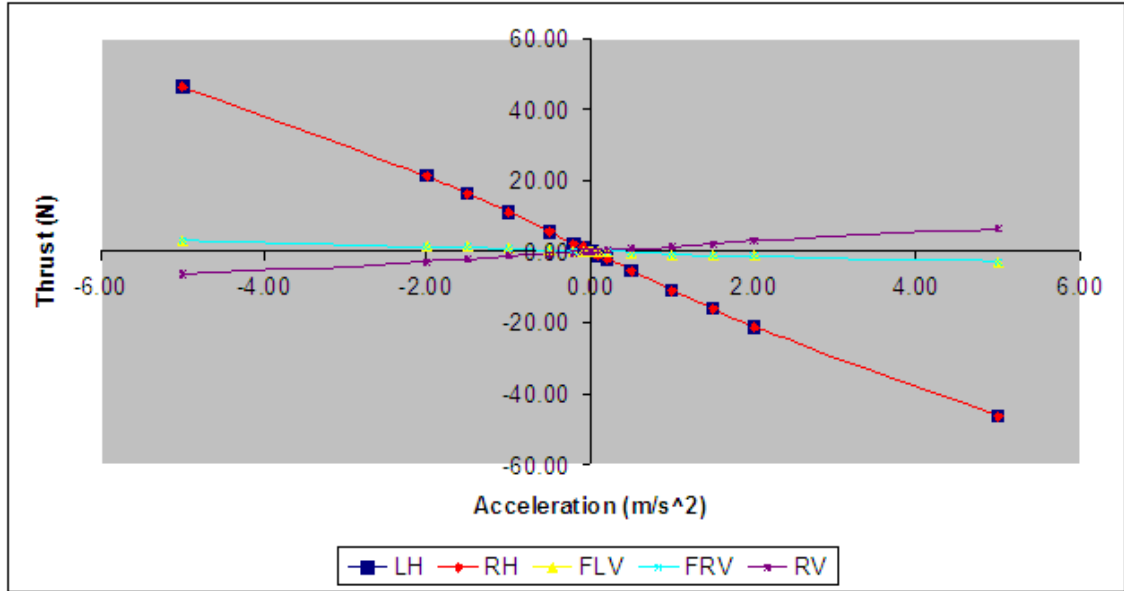


Figure 7.1: Acceleration in X direction

Examining Figure 7.1, it can be seen that as the acceleration increases in the positive x direction, the controller will increase the negative thrust in the left and right horizontal thrusters. This will act to counter the motion of the vehicle. The vehicle will, as a result of this thrust, also tilt forwards as the thrusters are below the centre of mass. The vertical thrusters are required to produce a small force to counter this effect. These effects demonstrate that the controller will attempt to nullify these forces and stabilise the AUV. A similar stabilising effect occurs in the Z direction and the output is shown in Figure 7.5

### 7.1.2 Angular Velocity Testing

The angular velocity values are updated by the MotionPak. The reaction force of the thrusters should be increasing as the angular velocity increases and should be in the opposing direction. Due to the alignment of the thrusters, it is possible to control the velocities in each of the angular axis. The output values should be of a lower magnitude than the acceleration values due to the dynamics of the vehicle. The results of testing performed on the angular velocity values are shown in Figures 7.2, 7.6 and 7.7.

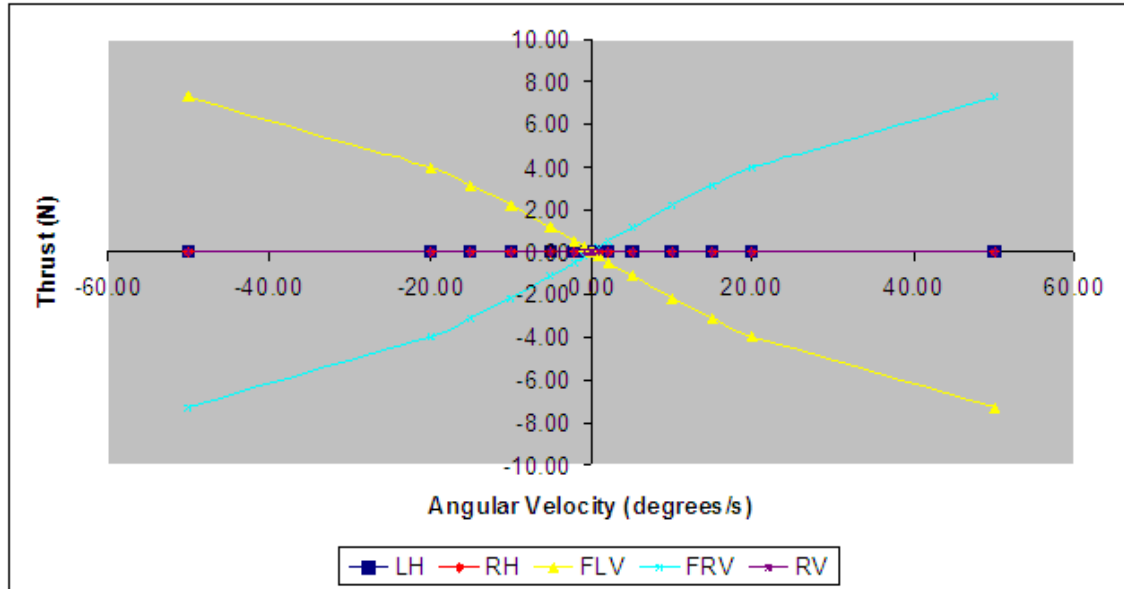


Figure 7.2: Angular Velocity in X direction

Figure 7.2 demonstrates that as the angular velocity increases in the positive direction around the X axis, the controller will increase the negative thrust in the front left vertical thruster and the positive thrust in the front right vertical thruster. This will act on the vehicle by opposing the motion and attempting to reduce this value to zero. There is no effect on the other thrusters as the front vertical thrusters are able to oppose the motion without causing additional effects to the vehicle. The magnitude of the resulting thrust is less than the acceleration values as expected, as the buoyancy of the vehicle will assist in cancelling this effect. These effects demonstrate that the controller will attempt to nullify these effects and stabilise the AUV. A similar stabilising effect occurs for the angular velocities in the Y and Z directions, where the output is shown in Figures 7.6 and 7.7

### 7.1.3 Position Testing

The position values which are updated by the sensors are the depth in Z direction and the angular displacements measured in Euler Angles. The depth is measured by the pressure sensor, while the inclinometers and compass measure the Euler Angles. The reaction force of the thrusters should be increasing as the displacement from the target value or reference value increases and should be in the opposing direction. Currently, there is no accurate way of measuring



the displacement in the X or Y direction, so these displacement values are not measured. The results of testing performed on the acceleration values are shown in Figures 7.3, 7.8, 7.9 and 7.10.

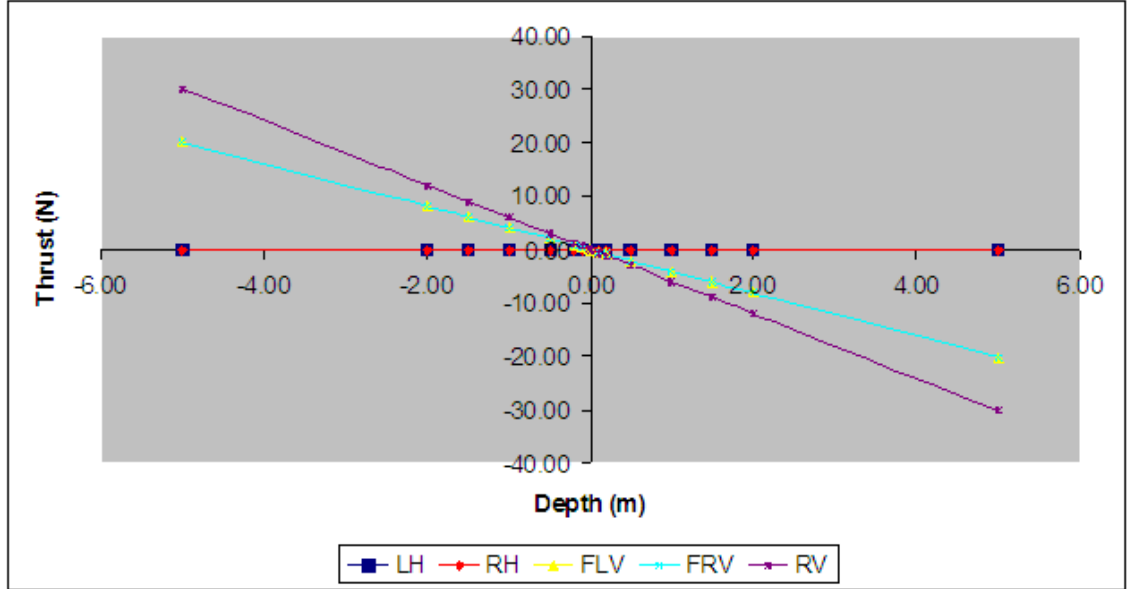


Figure 7.3: Displacement in Z direction

By examining Figure 7.3, it can be seen that as the depth increases in the positive Z direction, the controller will increase the negative thrust in the front left, front right and rear vertical thrusters. This will act to counter the motion of the vehicle. There is no effect on the other thrusters as the vertical thrusters are able to oppose the motion without causing additional effects to the vehicle. These effects demonstrate that the controller will attempt to reach the target reference value and stabilise the AUV.

A similar effect can be seen for the rotations around the X and Z axis in Figures 7.8 and 7.10. However, this implementation has been made using Euler Angles and the Y axis has an undefined effect due to the singularities, which can be seen in Figure 7.9. This demonstrates that the controller becomes unstable when the angle  $\theta = \pm\frac{\pi}{2}$  is approached. To address this issue, a conditional statement was included in the code and the corresponding output performed as expected, as shown in Figure 7.11.

#### 7.1.4 Overall Effects

The preceding results demonstrate that the controller is able to appropriately react to changes in the sensor readings to stabilise the motion of the AUV, across the full range of possible input values.

### 7.2 Real Time Integration

In order to ensure that the system will operate in the real time environment with precision, the implementation of the real time analysis specified in Chapter 6 was examined. These tests are intended to prove that the protected sensor object is able to effectively manage the sensors' readings and the controller task is able to quickly perform its mathematical operations.

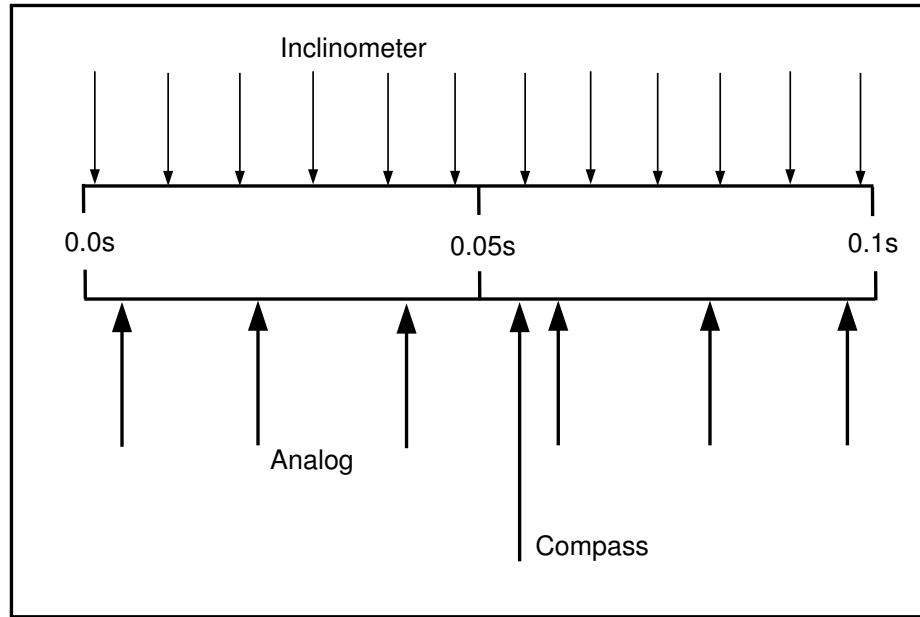
Initially the timing for the main controller task was taken to ensure that the operations could be performed in the specified time frame. On the testing machine, which had a 1Ghz processor, one full execution of the task was completed in 0.6 ms. Kambara has a Force CPCI-3740 processor which has a speed of 233 Mhz and theoretically allows Kambara to complete this task within 5ms.

A detailed test was performed to examine when the protected sensor object, as described in Chapter 6, gets accessed by the various tasks. The time was recorded when each task accessed the protected sensor object. The aim of the test was to ensure that the object gets updated at the frequencies which the sensors produce output, as specified in Chapter 3, without causing unpredictable synchronisation tasks. A timeline based on these results was created to demonstrate the operation of the system. The timeline can be seen in Figure 7.4. This figure does not include access by the controller task which retrieves the information from the protected object every 0.1 seconds.

These results confirm that the AUV is able to run in the real time environment. The sensors update their values as specified by the frequencies that they produce data. The controller accesses the protected object and is able to retrieve the most recent updated readings. Potentially the controller could be updated at a faster rate of almost 0.07s, being dependant on the slowest sensor. However if the output changes too frequently, the thruster controller will not be able to reach the value specified by the main controller.

### 7.3 Further Testing

Once it has been determined that the controller attempts to stabilise the system and that the controller can operate in the real time environment, real world



**Figure 7.4: Timeline, Representing Access to the Protected Sensor Object**

testing can begin. A pool built for the testing of Kambara is located at the rear of RSISE and is available for controlled testing of the submersible.

Unfortunately due to circumstances outside of my control, Kambara has not been available for pool testing for the duration of my time at RSISE. At the time of writing this thesis the low level code written to communicate between the CPU and the multiple cards operating on the PCI bus was not working. This vital communication enables access to the serial ports, the ADC channels and the IP68332 card which generates the PWM signals. As a result all of the sensors cannot be accessed on Kambara and the output cannot be sent to the amplifiers. The old software is to be replaced and the design and implementation tasks for this work could take up to two months to complete. The next step in progressing this project is to perform pool testing and gather results on the real world performance of the control system.

## 7.4 Possible Errors in the Controller

This control design is reliant on the system model accurately representing the hydrodynamic effects which will occur during operation of the AUV. The system model presented in Chapter 2, includes a number of terms to attempt to fully define these effects. However, a number of these values have been simplified and

have been determined experimentally, introducing inaccuracies to the system. The problem of inaccuracies in hydrodynamic effects has been investigated and has resulted in control approaches by other AUV's that are adaptive or robust to these effects [12]. A similar system model, as described by Silpa-Anan [16], was able to be used with the aid of vision to reliably control the vehicle in controlled pool testing. This demonstrates that the system model is capable of performing in the tank environment.

An additional source of errors will be generated by the inertial sensor suite. These sensors will all introduce errors to the system, through inaccuracies in measuring the required inertial data. These sensors are precision devices and the typical maximum error that is expected from the sensors is 1%, in readings from the pressure sensor. The remaining sensors in the suite produce outputs with errors of less than 1%. In addition these sensors all need to be calibrated and correctly aligned for operation, introducing additional errors to the measurements. If the sensors are correctly positioned and calibrated it is anticipated that the errors generated from the sensors will be insignificant in testing and will not prevent effective control of the AUV's motion.

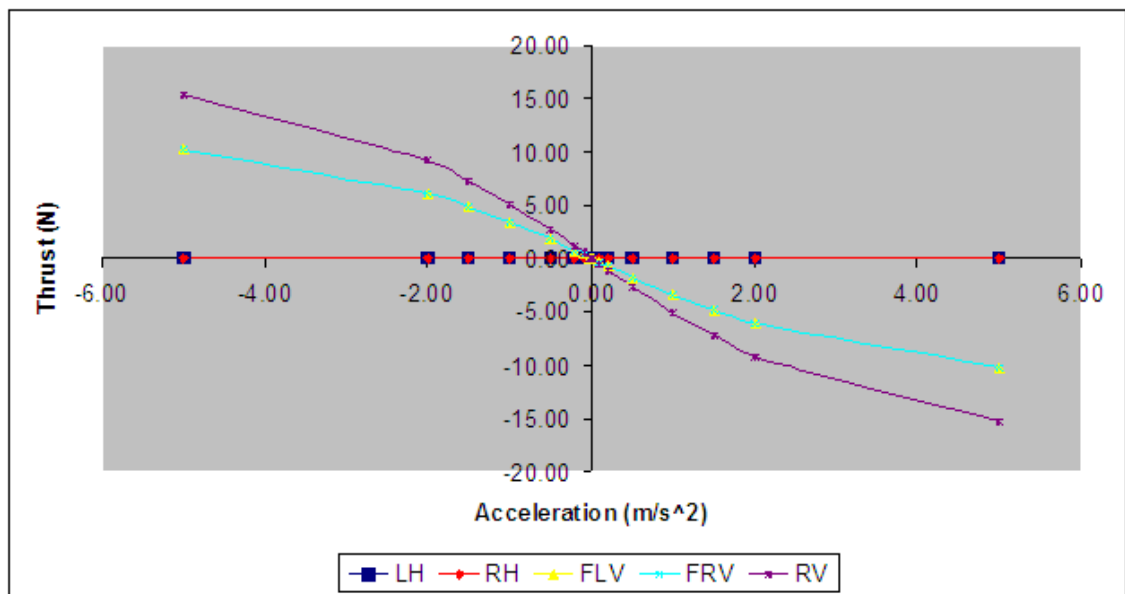


Figure 7.5: Acceleration in Z direction

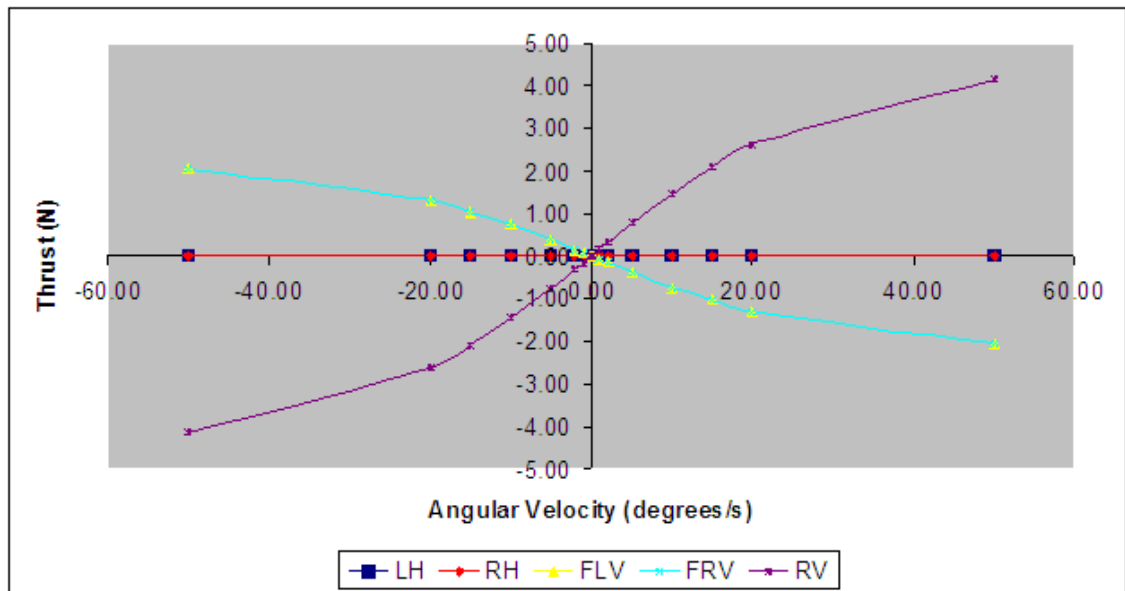


Figure 7.6: Angular Velocity in Y direction

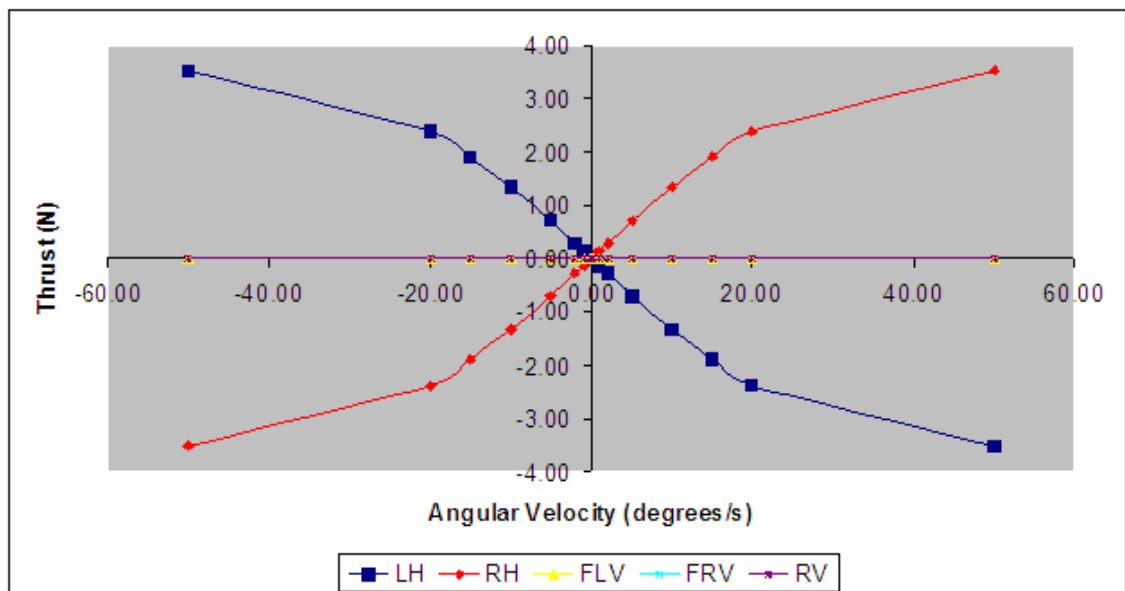


Figure 7.7: Angular Velocity in Z direction

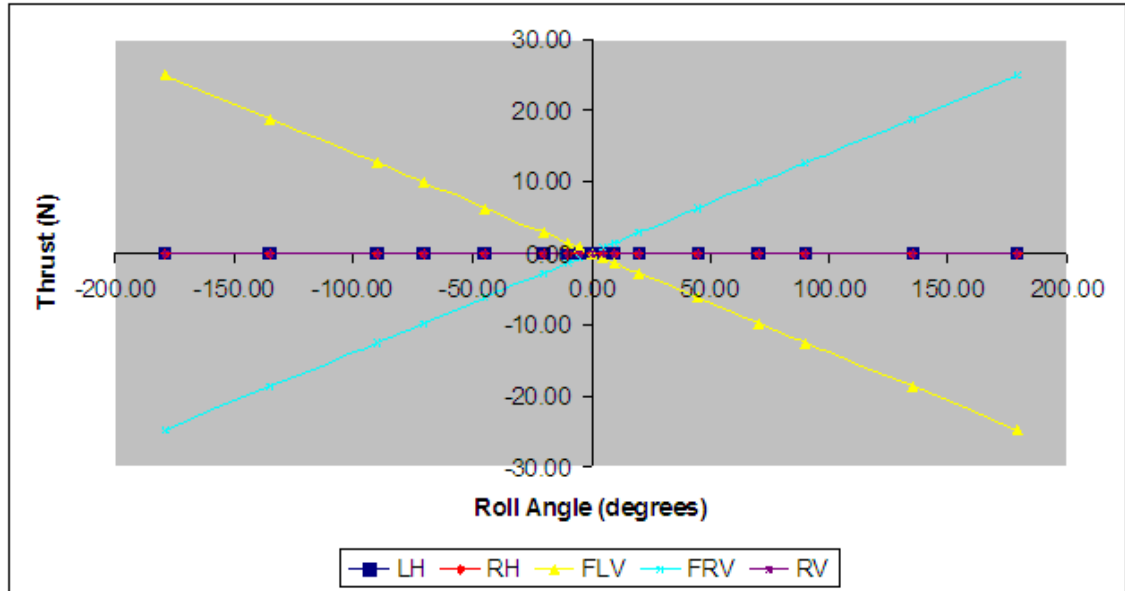


Figure 7.8: Angular Displacement in X direction

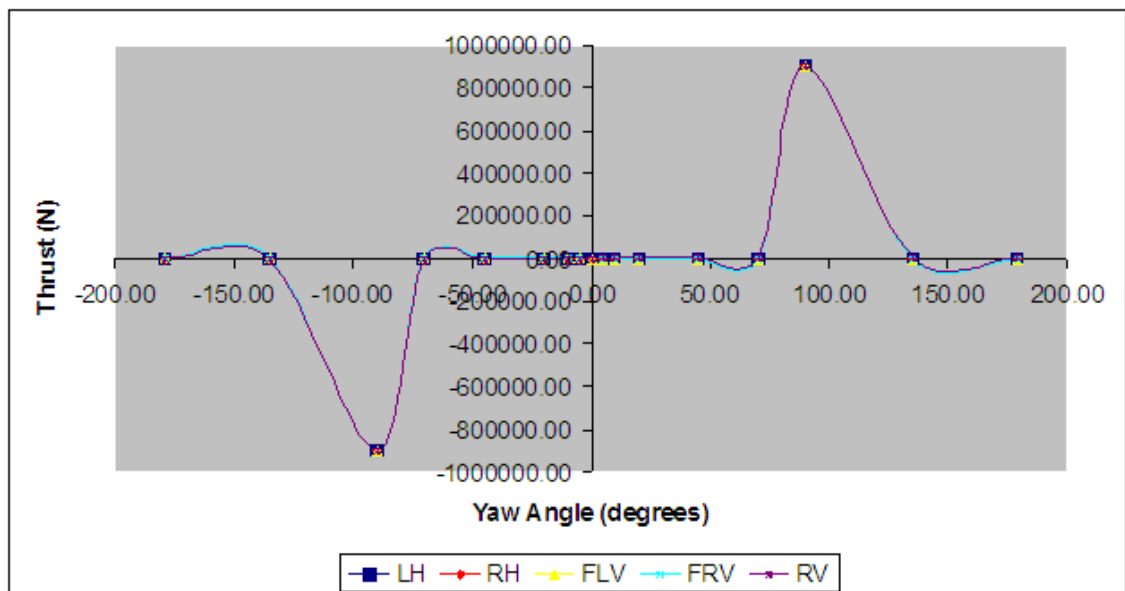


Figure 7.9: Angular Displacement in Y direction Showing Singularities

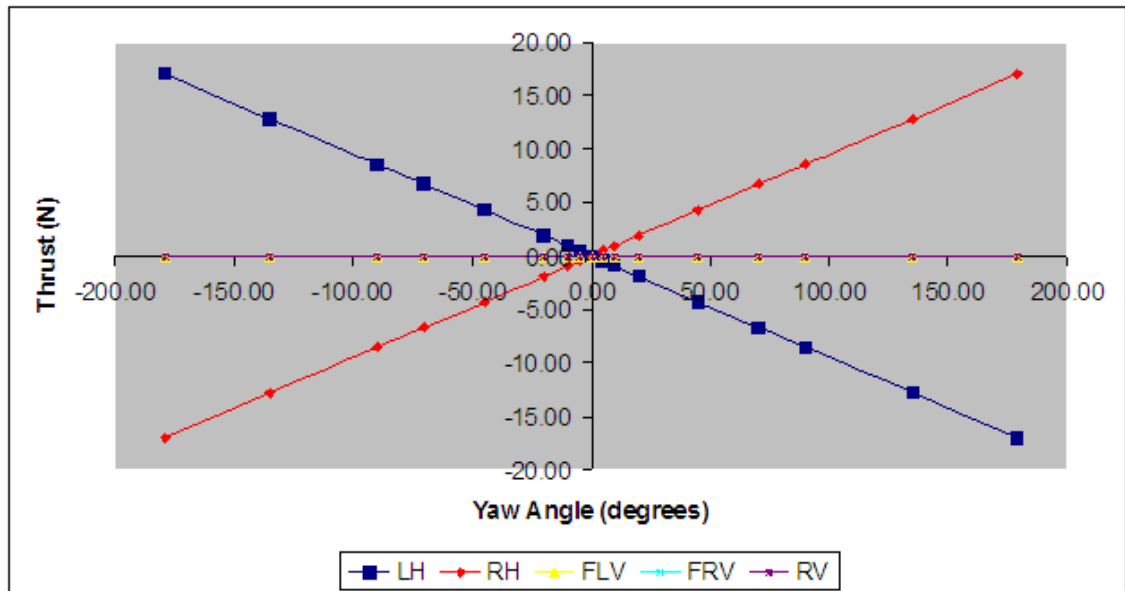


Figure 7.10: Angular Displacement in Z Direction

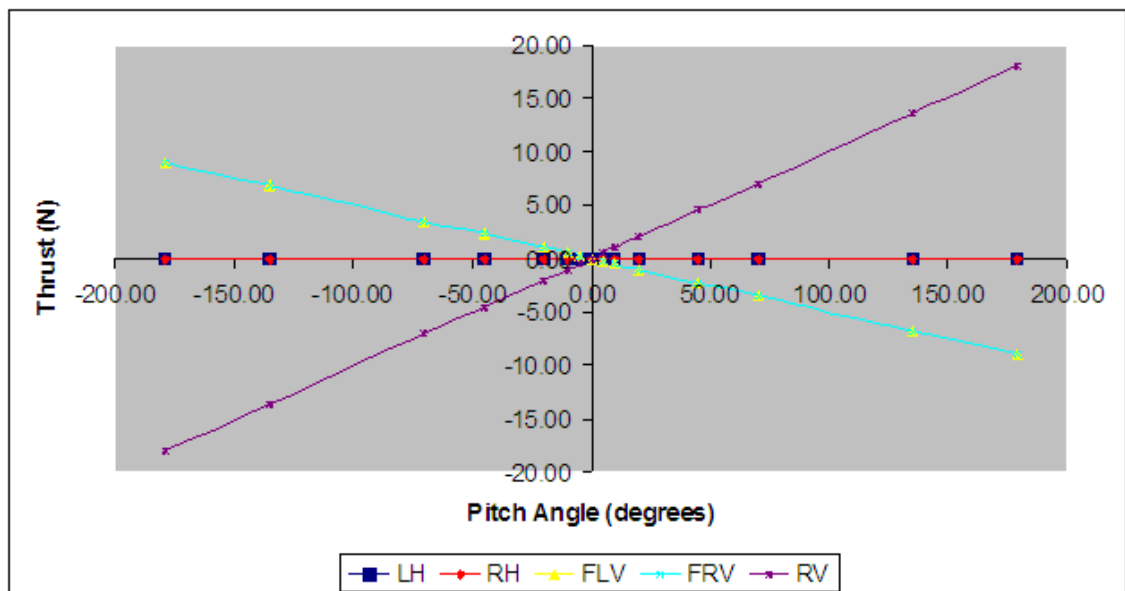


Figure 7.11: Non Singularity Angular Displacement in Y direction

---

# Conclusion

---

This thesis originally aimed to create a control system for the Kambara system. This design was to utilise the inertial sensors onboard Kambara and to demonstrate their real time capabilities in controlling the AUV. The implementation of the design was also to be fully tested to demonstrate the functionality of the AUV.

The work presented in this thesis has met all of the project objectives, excluding the final testing in the pool environment. A control system had been created based on a detailed system model that utilises the inertial sensors. The real time issues have been analysed and incorporated into the control design. The implementation of this design has been tested and demonstrates that it will stabilise Kambara in the six degrees of freedom underwater environment.

The final demonstration of the stability control for Kambara by physical testing in underwater environment has not been able to occur. This has been due to a low level software bug in previous versions of the code for Kambara. The next step to develop this project is to undertake pool testing to demonstrate the operation of the AUV. This will require the low level software to be reimplemented providing reliable access to all hardware components on the AUV.

Once the operation of the stability controller has been demonstrated on the AUV, additional work can be undertaken on the controller to enable it to track a reference velocity. This will enable the control of simple movements of the AUV in the underwater environment.

The tested velocity controller can then be used as a reliable basis for further work in terms of underwater guidance and mapping tasks. These tasks would provide the reference signal to the velocity controller enabling navigation of the underwater environments. Kambara contains a sonar sensor and cameras which can be used to provide information about the current environment of the AUV, to assist in this process.

Once Kambara can reliably control the velocity of the vehicle, a number of underwater tasks can be performed by the AUV. These tasks include observing



marine life, explorations into the deep sea and under ice-topped regions, and the inspection and maintenance of underwater structures and cabling. This work will have applications in the fields of marine biology, marine geology, underwater inspection and assistance. Research into the underwater environment will help us to understand more about this under utilised resource. Investigation could, for example, include the detailed study of the dispersal and colonisation of the sedentary fouling serpulid worm pictured in Figure 1.1. The biology of these marine pests could possibly, then be understood and a method of solving the problem they create could be achieved. Significant savings may be achievable in the cost of maintaining ship's hulls free of marine fouling organisms, also helping to limit the potential for translocation of organisms with invasive characteristics in different marine environments.

The work completed in this thesis acts as a solid basis for further development of the inertial control of Kambara. Further development on this control design will assist in the evolution of the AUV concept and the potential for realising future applications.

---

# Bibliography

---

- [1] Anonymous. An introduction to ada. [http://www.adahome.com/ Discover/Introduction.html](http://www.adahome.com/Discover/Introduction.html), 1995.
- [2] Martin Baker. Conversion euler to axis-angle. 2003.
- [3] D. Beswick. Inertial navigation system for an autonomous underwater vehicle. Technical report, Australian National University, 1998.
- [4] T. Betlehem. Autonomous submersible robot: State estimation system. Technical report, Australian National University, 1999.
- [5] A. Burns and A. Wellings. *Concurrency in Ada*. Cambridge, second edition, 1998.
- [6] N. H. Cohen. *Ada as a Second Language*. McGraw Hall, second edition, 1996.
- [7] John Craig. *Introduction to Robotics Mechanics and Control*. Addison Wesley Publishing Company, second edition, 1986.
- [8] J. K. Cvetanovski. Autonomous submersible robot: Sensor characterisation and testing. Technical report, Australian National University, 2000.
- [9] O. Fjellstad and T. I. Fossen. Position and attitude tracking of auvs: A quaternion feedback approach. *IEEE Journal of Oceanic Engineering*, 19, 1994.
- [10] T. I. Fossen. Underwater vehicle dynamics. Technical report, University of Trondheim, 1994.
- [11] Powell Franklin and Emani-Naeini. *Feedback Control of Dynamic Systems*. Prentice Hall, fourth edition, 2002.
- [12] N. Sarker G. Antonelli, S. Chiaverine and M. West. Adaptive control of an autonomous underwater vehicle: Experimental results on odin. *IEEE Transactions on Control Systems Technology*, 9, September 2001.
- [13] CSIRO Marine Research. Marine pest information sheet - hydroides sanctae-cruis. *CRIMP Infosheet 15*, 2001.

- 
- [14] Wind River. Wind river powers mars exploration rovers: Continues legacy as technology provider for nasa's space exploration. <http://europe.windriver.com/news/pressrelease.html?ID=100287>, 2003.
  - [15] J. Rosenblatt G. Dissanayake S. Williams, P. Newman and H. Durrant-Whyte. Autonomous underwater navigation and control. Technical report, University of Sydney, 2001.
  - [16] Chanop Silpa-Anan. *Visual Control of an Autonomous Underwater Vehicle*. PhD thesis, Australian National University, 2001.
  - [17] S. Sukkarieh. Experimental results on the minnkota classic series motor. Technical report.
  - [18] S. Williams, S. Majumder, J. Rosenblatt, and H. Durrant-Whyte. Autonomous transect surveying of the great barrier reef. In *Australian Conference on Robotics and Automation*, 1999.
  - [19] S. L. Wood. Underwater technologies laboratory. <http://www.fit.edu/AcadRes/dmes/subsea.html>, 2003.

# MotionPak Data Sheet

**BEI** SYSTRON DONNER INERTIAL DIVISION  
SENSORS & SYSTEMS COMPANY

A Subsidiary of BEI TECHNOLOGIES, INC.

2700 Syston Drive • Concord, CA 94518-1399 • Phone: (925) 682-6161 • Fax: (925) 671-6590

**MotionPak**  
Final Test Data Sheet

Customer: **BEI SENSORS & SYSTEMS CO.**

Model No.: **MP-GPDD00BBR-100**

Serial No.: **0499**

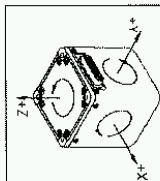
Date: **1/11/99**

Input Current (+15 Vdc): **231  $\mu$ A**

Input Current (-15 Vdc): **-186  $\mu$ A**

Package Weight: **892.071**

Temp Sensor (AD590): **11A $^{\circ}$ K**



	ANGULAR X-AXIS	ANGULAR Y-AXIS	ANGULAR Z-AXIS	LINEAR X-AXIS	LINEAR Y-AXIS	LINEAR Z-AXIS
Range	100 $\mu$ s	100 $\mu$ s	100 $\mu$ s	2 g	2 g	2 g
Scale Factor	24.804 mV/ $\mu$ s	24.922 mV/ $\mu$ s	24.943 mV/ $\mu$ s	3.753 V/g	3.747 V/g	3.760 V/g
SF Temp Performance	< $\pm 0.03$ %/ $^{\circ}$ C	< $\pm 0.03$ %/ $^{\circ}$ C	< $\pm 0.03$ %/ $^{\circ}$ C	< $\pm 0.03$ %/ $^{\circ}$ C	< $\pm 0.03$ %/ $^{\circ}$ C	< $\pm 0.03$ %/ $^{\circ}$ C
Bias (@ +22 $^{\circ}$ C)	0.10 $\mu$ s	+0.01 $\mu$ s	0.07 $\mu$ s	-2.52 mg	-3.76 mg	-1.93 mg
Bias Temp Performance	< 3%/stem $\pm 22^{\circ}$ C	< 3%/stem $\pm 22^{\circ}$ C	< 3%/stem $\pm 22^{\circ}$ C	< 100 $\mu$ g/ $^{\circ}$ C	< 100 $\mu$ g/ $^{\circ}$ C	< 100 $\mu$ g/ $^{\circ}$ C
Alignment	0.36 $^{\circ}$	0.22 $^{\circ}$	0.72 $^{\circ}$	0.12 $^{\circ}$	0.14 $^{\circ}$	0.04 $^{\circ}$
Bandwidth (<90 $^{\circ}$ )	>60 Hz	>60 Hz	>60 Hz	>300 Hz	>300 Hz	>300 Hz
Noise (10-100 Hz)	< 0.01 $^{\circ}$ /sec/ $\sqrt{\text{Hz}}$	< 0.01 $^{\circ}$ /sec/ $\sqrt{\text{Hz}}$	< 0.01 $^{\circ}$ /sec/ $\sqrt{\text{Hz}}$	$\leq 7.0$ mVRMS	$\leq 7.0$ mVRMS	$\leq 7.0$ mVRMS

\*Low Noise (30.005  $^{\circ}$ /sec/ $\sqrt{\text{Hz}}$ ) may be specified per customer request. See specific model ICD.

Test Data Reviewed by: *Mac Riddick*

CD1208 REV. A

For Customer Service Phone: (925) 671-6464 or E-mail: service@systron.com

Figure A.1: Manufacturer's data sheet for the MotionPak

---

# Thruster Output

---

The output of the thrusters was obtained experimentally by S. Sukkariéh[17]. They show the output response of the thrusters in both the forward and reverse directions, and with and without the duct on.

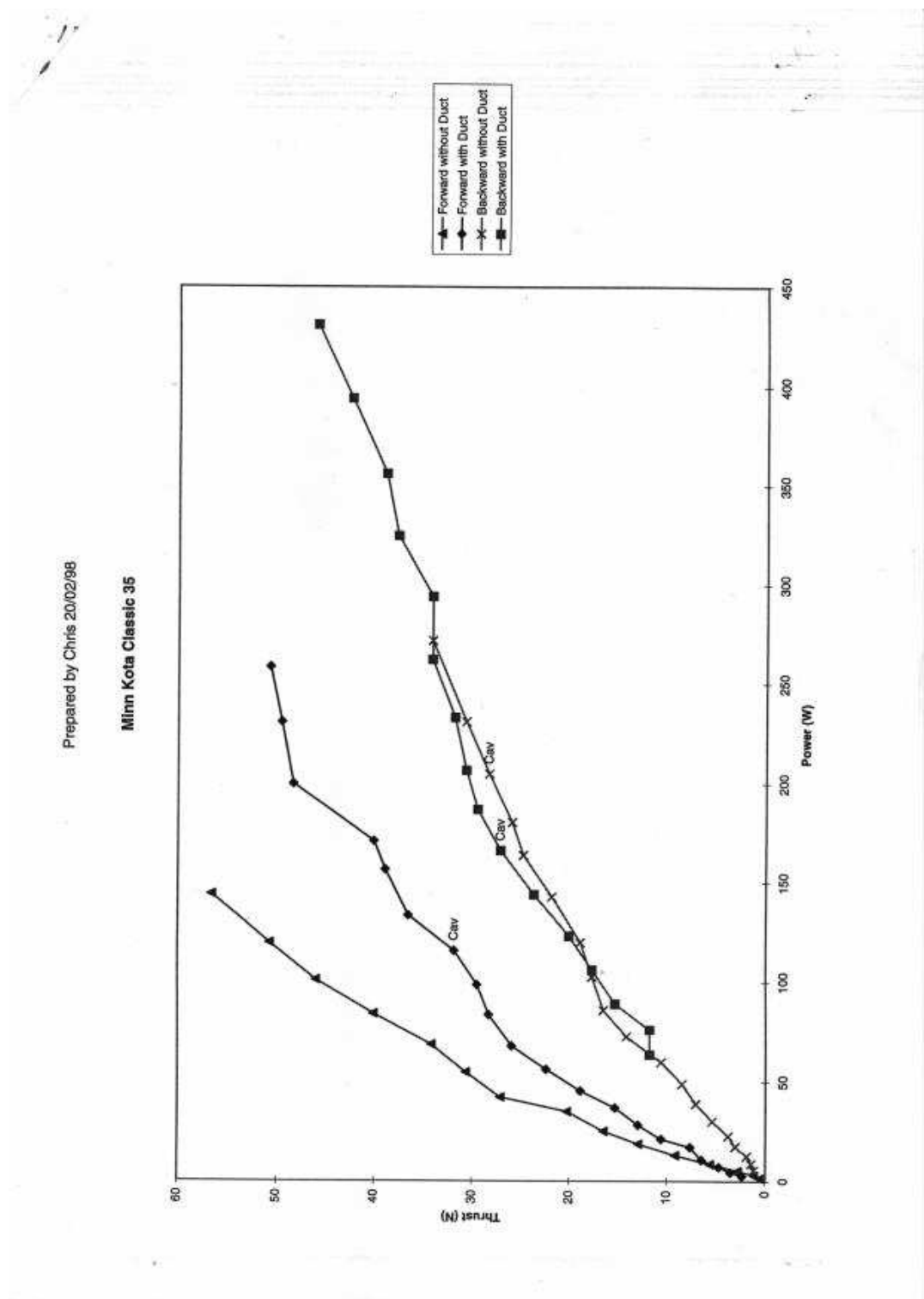


Figure B.1: Minnkota Thruster Output Results

---

# Depth Sensor Testing

---

The results of testing performed by Beswick [3] produced the results shown in Figure C.1.

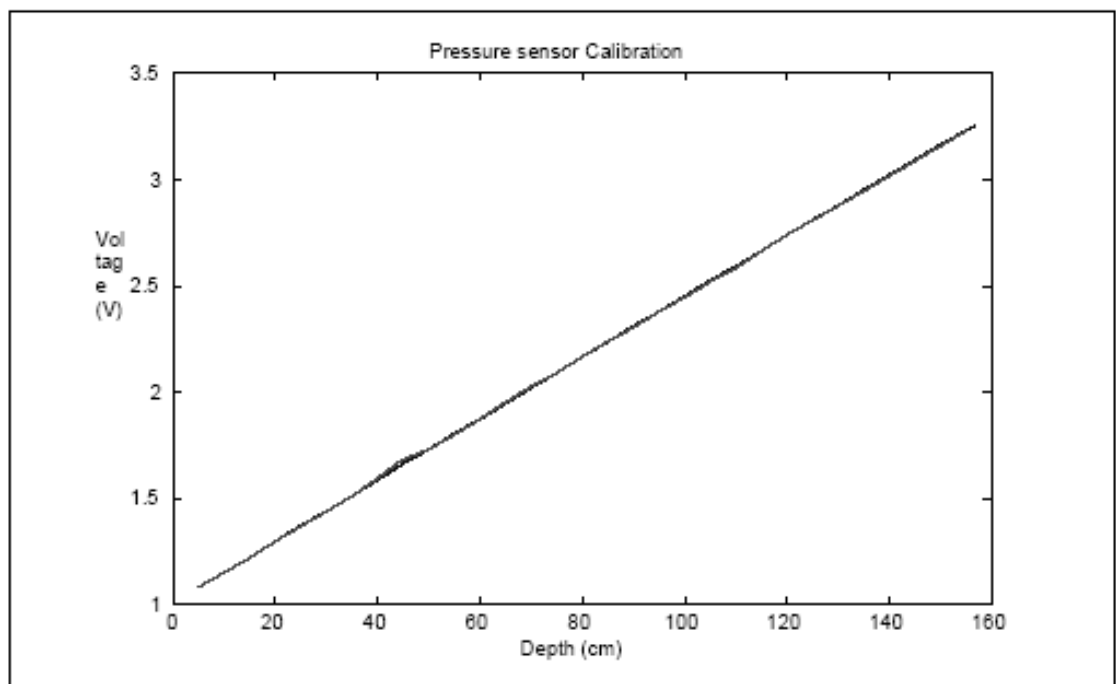


Figure C.1: Pressure Sensor Output

The linear output is 14mV/cm. Additional testing by Beswick [3] on a smaller depth scale demonstrated (See Figure C.2) that the sensor is sensitive to depth changes of 0.5 cm.

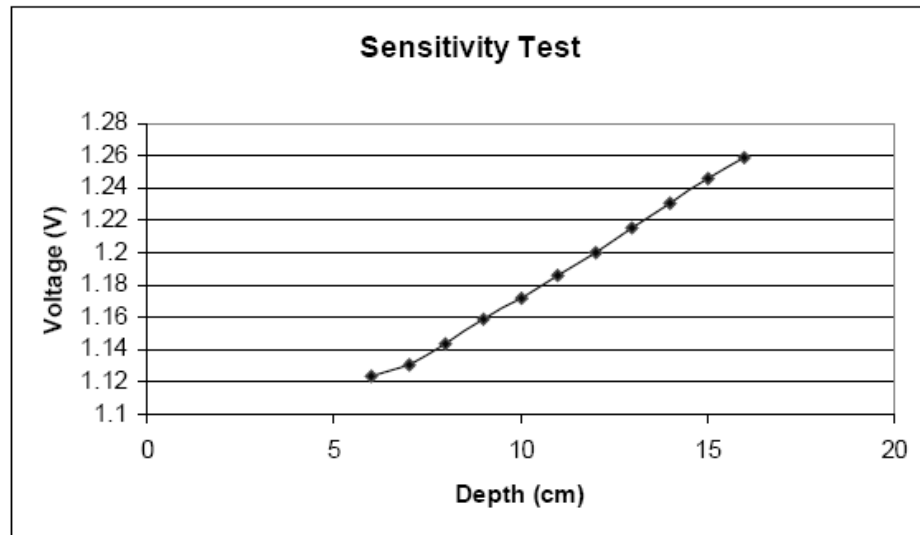


Figure C.2: Sensitivity Testing of Pressure Sensor



---

# Ada Controller Code

---

The controller code shown below includes the following files:

- Main3.adb;
- Protected\_Sensor\_Data\_Package.ads; and
- Protected\_Sensor\_Data\_Package.adb.

```
--
--                                     -*- Mode: Ada -*-
-- Filename      : Main3.adb
-- Description    : Main routine for the Controller of Kambara.
-- Author        : David Biddle
-- Created On     : Wed 5th March 2003
-- Last Modified By: David
-- Last Modified On: Mon 11-06-03
-- Update Count   : 4
-- Status        : Under Development

with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
with Protected_Sensor_Data_Package; use Protected_Sensor_Data_Package;
with System.Storage_Elements; use System.Storage_Elements;
with Ada.Float_Text_IO; use Ada.Float_Text_IO;
with Ada.Numerics.Elementary_Functions; use Ada.Numerics.Elementary_Functions;
with Ada.Real_Time; use Ada.Real_Time;

procedure Main3 is
  task Controller;
  task Serial_Inclinometer;
  task Serial_Compass;
  task Analog_Sensors;

  task body Controller is

    type Float_Array is array (1 .. 6) of float;
    type Thrust_Array is array (1 .. 5) of float;
```

---

```

type Short_Float_Array is array (1 .. 3) of float;
type Matrix is array (1..3, 1..3) of Float;

--Special testing type
type Long_Float_Array is array (1 .. 17) of float;

Zero_Vector      : Float_Array := (others => 0.0);
Short_Zero_Vector : Short_Float_Array := (others => 0.0);

Sensor_Data : Protected_Sensor_Data_Package.Sensor_Data_Type := (others => 0.0);

-- Reference values (what we are trying to achieve)
R : Float_Array := Zero_Vector;

-- Cos and Sine values of the angular sensor readings
C : Short_Float_Array := Short_Zero_Vector ;
S : Short_Float_Array := Short_Zero_Vector ;

-- Rotational Matrix ensuring the values form the MotionPak are
-- referenced to the world reference frame
M,N : Matrix;

-- World Acceleration Value
Acc : Float_Array := Zero_Vector;

-- A constant vector used in determining thrust
A : Float_Array := Zero_Vector;

-- Current World Angular Velocity values
Odot, Odot2 : Short_Float_Array := Short_Zero_Vector;

-- Current World Velocity value
U : Short_Float_Array := Short_Zero_Vector;
-- Previous orld Velocity Value
V : Short_Float_Array := Short_Zero_Vector;

-- Current Approximate Displacement Value from initial position
Dis : Float_Array := Zero_Vector;
-- Previous Approximate Displacement Value from initial position
Dis_Last : Float_Array := Zero_Vector;

-- The sensor readings to compare to the reference signal
Sensor : Float_Array := Zero_Vector;

-- Interim results to help determine the total thrust
B : Float_Array := Zero_Vector;

-- Error signal, difference between the reference and signal values
E : Float_Array := Zero_Vector;

```

---

```

-- Gravitational and buoyancy forces placed on Kambara
G : Float_Array := Zero_Vector;

-- Interim results to help determine the total thrust
L : Float_Array := Zero_Vector;

-- Thrust required to control Kambara
T : Float_Array := Zero_Vector;

-- Required thrust for the five thrusters
Output_Thrust: Thrust_Array := (1 => 0.0, 2 => 0.0, 3 => 0.0, 4 => 0.0, 5 =>0.0);

Time_Period, Time_Sq, D : float := 0.0;

-- Control Variables
K1,K2 : float := 0.0;

-- Sensor variables
D_dash : float := 0.0;
O : Short_Float_Array := Short_Zero_Vector;
A_dash : Short_Float_Array := Short_Zero_Vector;
Odot_dash : Short_Float_Array := Short_Zero_Vector;

X : constant Integer :=1;
Y : Constant Integer :=2;
Z : constant Integer :=3;

Tf,Tf2    : Time;
Nano1     : Time_Span;
Timeval   : Integer;
Real_Time : Duration;

begin

--Wait for other sensors to initialise
delay 0.2;
Protected_Sensor_Data.Set_Start;

Tf := Clock;
loop
    Sensor_Data := Protected_Sensor_Data.Get_Sensor_Data;

    O(X) := Sensor_Data(1);
    O(Y) := Sensor_Data(2);
    O(Z) := Sensor_Data(3);

    D_Dash := Sensor_Data(4);

```

---

```

A_Dash(x) := Sensor_Data(5);
A_Dash(y) := Sensor_Data(6);
A_Dash(z) := Sensor_Data(7);

-- Obtain and normalised angular values from degrees per sec
Odot_Dash(x) := Sensor_Data(8)/10.0;
Odot_Dash(y) := Sensor_Data(9)/10.0;
Odot_Dash(z) := Sensor_Data(10)/10.0;

-- Time Period is how long each iteration of entire control loop takes
Time_Period := 0.1;

-- User Defined reference values
R(1) := 0.0;
R(2) := 0.0;
R(3) := 1.5;
R(4) := 0.0;
R(5) := 0.0;
R(6) := 0.0;

-- The angles for the rotation matrix M
C(x) := Cos(O(x), 360.0);
C(y) := Cos(O(y), 360.0);
C(z) := Cos(O(z), 360.0);
S(x) := Sin(O(x), 360.0);
S(y) := Sin(O(y), 360.0);
S(z) := Sin(O(z), 360.0);

-- Simplified rotation matrix
M(1,1) := C(z)*C(y);
M(1,2) := -1.0*S(z)*C(x) + S(x)*C(z)*S(y);
M(1,3) := S(z)*S(x) + C(z)*S(y)*C(x);
M(2,1) := S(z)*C(y);
M(2,2) := C(z)*C(x) + S(z)*S(y)*S(x);
M(2,3) := -1.0*S(x)*C(z) + S(z)*S(y)*C(x);
M(3,1) := -1.0*S(y);
M(3,2) := C(y)*S(x);
M(3,3) := C(y)*C(x);

-- Rotate the Acceleration and Angular Velocity values to ensure in
-- parallel to world reference frame

if (((O(y) < 100.0) and (O(y) > 80.0)) or ((O(y) < 280.0) and (O(y) > 260.0))) then
    -- Allow for Singularities in Euler Angles for 80.0 to 100 degrees

    if O(y) > 180.0 then

```

---

```

        C(y) := Cos(260.0, 360.0);
    else
        C(y) := Cos(80.0, 360.0);
    end if;

    N(1,1) := 1.0;
    N(1,2) := (S(y)*S(x))/C(y);
    N(1,3) := (S(y)*C(x))/C(y);
    N(2,1) := 0.0;
    N(2,2) := C(x);
    N(2,3) := -1.0*S(x);
    N(3,1) := 0.0;
    N(3,2) := S(x)/C(y);
    N(3,3) := C(X)/C(y);

    for I in X..Z loop
        for J in X..Z loop
            Odot(I) := Odot(I) + N(I,J)*Odot_dash(J);
        end loop;
    end loop;

    if O(y) > 180.0 then
        C(y) := Cos(280.0, 360.0);
    else
        C(y) := Cos(100.0, 360.0);
    end if;

    N(1,1) := 1.0;
    N(1,2) := (S(y)*S(x))/C(y);
    N(1,3) := (S(y)*C(x))/C(y);
    N(2,1) := 0.0;
    N(2,2) := C(x);
    N(2,3) := -1.0*S(x);
    N(3,1) := 0.0;
    N(3,2) := S(x)/C(y);
    N(3,3) := C(X)/C(y);

    for I in X..Z loop
        for J in X..Z loop
            Odot2(I) := Odot(I) + N(I,J)*Odot_dash(J);
            Acc(I) := Acc(I) + M(I,J)*A_Dash(J);
        end loop;
    end loop;

-- Determine the linear approximation of the value close to the singularities
if O(y) > 180.0 then
    for I in X..Z loop
        Odot(I) := ((280.0 - O(y))/20.0) * Odot(I) + ((O(y) - 260.0)/20.0) * Odot2(I);
    end loop;

```

---

```

else
    for I in X..Z loop
        Odot(I) := ((100.0 - O(y))/20.0) * Odot(I) + ((O(y) - 80.0)/20.0) * Odot2(I);
    end loop;
end if;

else
    -- As Y axis angle is not close to singularities perform normal calculations

    N(1,1) := 1.0;
    N(1,2) := (S(y)*S(x))/C(y);
    N(1,3) := (S(y)*C(x))/C(y);
    N(2,1) := 0.0;
    N(2,2) := C(x);
    N(2,3) := -1.0*S(x);
    N(3,1) := 0.0;
    N(3,2) := S(x)/C(y);
    N(3,3) := C(X)/C(y);

    for I in X..Z loop
        Acc(I) := 0.0;
        Odot(I) := 0.0;
        -- Clean V and U for testing purposes
        -- V(I) := 0.0;
        -- U(I):=0.0;
    end loop;

    for I in X..Z loop
        for J in X..Z loop
            Acc(I) := Acc(I) + M(I,J)*A_Dash(J);
            Odot(I) := Odot(I) + N(I,J)*Odot_dash(J);
        end loop;
    end loop;
end if;

-- Determine the velocity and acceleration. Making assumption
-- that the acceleration is constant over the time period

Time_Sq := Time_Period*Time_Period;

for I in X..Z loop
    V(I):= Acc(I) * Time_Period + U(I);
    Dis(I) := (Acc(I)/2.0)*Time_sq + U(I)*Time_Period + Dis_Last(I);
end loop;

for I in X..Z loop
    -- Normalise the value of the Velocity

```

---

```

    U(I) := V(I);
    Dis_last(I) := Dis(I) ;
end loop;

-- Adjust the depth value to ensure referenced to the centre of the submarine
-- Sensor suite is away from the centre of the sub
D := D_Dash;

-- Actual Sensor Values
Sensor(1) := 0.0;
Sensor(2) := 0.0;
Sensor(3) := D;
Sensor(4) := 0(x);
Sensor(5) := 0(y);
Sensor(6) := 0(z);

-- Solve  $T = G + B(\dot{X})\dot{X} - A(K_1\dot{x} + K_2(x-R))$ 

-- B( $\dot{X}$ ) $\dot{X}$  values
B(1) := ((120.0 + 90.0*(abs V(x))) * V(x)) + 23.8*V(z)*0dot(y) - 23.8*V(y)*0dot(z);
B(2) := ((90.0 + 90.0*(abs V(y))) * V(y)) - 23.8*V(z)*0dot(x) + 58.4*V(x)*0dot(z);
B(3) := ((150.0 + 120.0*(abs V(z))) * V(z)) + 23.8*V(y)*0dot(x) - 58.4*V(x)*0dot(y);
B(4) := ((15.0 + 10.0*(abs 0dot(x))) * 0dot(x)) + 1.49*0dot(z)*0dot(y);
B(5) := 34.6*V(x)*V(z) + ((15.0 + 12.0*(abs 0dot(y))) * 0dot(y)) + 0.71*0dot(x)*0dot(z);
B(6) := -34.6*V(x)*V(y) + ((18.0 + 15.0*(abs 0dot(z))) * 0dot(z)) - 2.2*0dot(x)*0dot(y);

-- A values
-- A = Ma + Mrb; Diagonal Matrix
A(1) := 175.4;
A(2) := 140.8;
A(3) := 140.8;
A(4) := 14.08;
A(5) := 13.6;
A(6) := 16.07;

-- For the control variables. Which are constant in execution
-- K1 := 10.0;
-- K2 := 0.1;
K1 := 2.0;
K2 := 1.0;

-- e := Sensor - R;
E(1) := Sensor(1) - R(1);
E(2) := Sensor(2) - R(2);
E(3) := Sensor(3) - R(3);
E(4) := Sensor(4) - R(4);
E(5) := Sensor(5) - R(5);
E(6) := Sensor(6) - R(6);

```

---

```

-- Adjust Angular error to ensure in the normalised range
for I in 4..6 loop

    if E(I) <= -180.0 then
        E(I) := 360.0 + E(I);
    elsif E(I) >= 180.0 then
        E(I) := 360.0 - E(I);
    end if;
    E(I) := E(I)/180.0;

end loop;

-- Adjust Depth error to ensure in the normalised range
E(3) := E(3)/10.0;

-- Gravitational and Buoyancy force values
-- The Weight and buoyancy of the sub has changed so this
-- value should be remeasured

-- Assume that this will always be approximately zero.
-- Future attempts to control Kambara will begin with
-- balancing the sub to ensure that no stationary forces
-- will occur. These values will need to be updated in this
-- is not the case.
G(1) := 0.0;
G(2) := 0.0;
G(3) := 0.0;
G(4) := 0.0;
G(5) := 0.0;
G(6) := 0.0;

--  $L = A(K1\dot{x} + K2(x-R))$ 

L(1) := A(1) * ( K1 * V(x) + K2*E(1));
L(2) := A(2) * ( K1 * V(y) + K2*E(2));
L(3) := A(3) * ( K1 * V(z) + K2*E(3));
L(4) := A(4) * ( K1 * Odot(x) + K2*E(4));
L(5) := A(5) * ( K1 * Odot(y) + K2*E(5));
L(6) := A(6) * ( K1 * Odot(z) + K2*E(6));

-- Torque required by thrusters in the 6 defined axis
T(1) := G(1) + B(1) - L(1);
T(2) := G(2) + B(2) - L(2);
T(3) := G(3) + B(3) - L(3);
T(4) := G(4) + B(4) - L(4);
T(5) := G(5) + B(5) - L(5);
T(6) := G(6) + B(6) - L(6);

```



---

```

-- Determine output values to thrusters by translating the
-- values so they can be produced by the 5 thrusters.

-- left_horizontal
Output_Thrust(1) := 0.5 * T(1) + 1.0638 * T(6);
-- right_horizontal
Output_Thrust(2) := 0.5 * T(1) - 1.0638 * T(6);
-- front_left_verticle
Output_Thrust(3) := 0.0 + 0.0333*T(1) + 0.2867*T(3) + 1.7857*T(4) + 0.6667*T(5);
-- front_right_verticle
Output_Thrust(4) := 0.0 + 0.0333*T(1) + 0.2867*T(3) - 1.7857*T(4) + 0.6667*T(5);
-- rear_verticle
Output_Thrust(5) := -0.0667*T(1) + 0.4267*T(3) - 1.3333*T(5);

New_Line;
Put_Line( "Thrust needed from the Left Horizontal" );
Ada.Float_Text_IO.Put(Output_Thrust(1));
New_Line;

Put_Line( "Thrust needed from the Right Horizontal" );
Ada.Float_Text_IO.Put(Output_Thrust(2));
New_Line;

Put_Line( "Thrust needed from the Front Left Verticle" );
Ada.Float_Text_IO.Put(Output_Thrust(3));
New_Line;

Put_Line( "Thrust needed from the Front Right Verticle" );
Ada.Float_Text_IO.Put(Output_Thrust(4));
New_Line;

Put_Line( "Thrust needed from the Rear Verticle" );
Ada.Float_Text_IO.Put(Output_Thrust(5));
New_Line;

-- Send Output_Thrust values to the thrusters
Nano1 := Nanoseconds(1);

tf2 := Clock;
Timeval := (Tf2-TF) / Nano1;
Real_Time := (Timeval) * 0.000000001;

delay ( 0.1 - Real_Time);
tf := Clock;
-- End of the infinite loop
end loop;
end Controller;
```

---

```

task body Serial_Inclinometer is
    -- Once the Serial Card is fully operational these values will be
    -- obtained from the sensors each iteration.
    Roll_Angle,Pitch_Angle : Float:=0.0;
begin
    loop
        delay 0.008;
        Roll_Angle:=0.0;
        Protected_Sensor_Data.Set_Roll_Angle(Roll_Angle);
        -- 8ms rounded up
        delay 0.008;
        Pitch_Angle:= 260.0;
        Protected_Sensor_Data.Set_Pitch_Angle(Pitch_Angle);
    end loop;
end Serial_Inclinometer;

task body Serial_Compass is
    -- Once the Serial Card is fully operational this value will be
    -- obtained from the sensor each iteration.
    Yaw_Angle : Float:=0.0;
begin
    loop

        -- 16 Hz frequency of the compass data
        delay 0.0625;
        Yaw_Angle:=0.0;
        Protected_Sensor_Data.Set_Yaw_Angle(Yaw_Angle);

    end loop;
end Serial_Compass;

task body Analog_Sensors is
    -- Once the ADC Card is fully operational these values will be
    -- obtained from the sensors each iteration.
    Depth,
    Acc_X,          Acc_Y,          Acc_Z,
    Ang_Velocity_X, Ang_Velocity_Y , Ang_Velocity_Z: Float:=0.0;
begin
    loop

        -- Update every 60 Hz slowest speed of Motion Pak
        delay 0.0167;

        Depth := 1.5;
        Acc_X := 0.0;
        Acc_Y := 0.0;
        Acc_Z := 0.0;
        Ang_Velocity_X:=0.0;
        Ang_Velocity_Y:=0.0;
    end loop;
end Analog_Sensors;

```

---

```

        Ang_Velocity_Z:=0.0;

        Protected_Sensor_Data.Set_Depth(Depth);
        Protected_Sensor_Data.Set_Acc_X(Acc_X);
        Protected_Sensor_Data.Set_Acc_Y(Acc_Y);
        Protected_Sensor_Data.Set_Acc_Z(Acc_Z);
        Protected_Sensor_Data.Set_Ang_Velocity_X(Ang_Velocity_X);
        Protected_Sensor_Data.Set_Ang_Velocity_Y(Ang_Velocity_Y);
        Protected_Sensor_Data.Set_Ang_Velocity_Z(Ang_Velocity_Z);

    end loop;
end Analog_Sensors;

begin
    null;
end Main3;

--
--          -- Mode: Ada --
-- Filename      : Protected_Sensor_Data_Package.ads
-- Description    : Most recent copy of sensor values
-- Author        : David Biddle
-- Created On    : 18th May 2003
-- Last Modified By: David Biddle
-- Last Modified On: .
-- Update Count  : 2
-- Status        : Under Construction

with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
with Ada.Float_Text_IO; use Ada.Float_Text_IO;
with Ada.Real_Time; use Ada.Real_Time;

package Protected_Sensor_Data_Package is

    type Sensor_Data_Type is array (1 .. 10) of float;

protected Protected_Sensor_Data is

    procedure Set_Roll_Angle(Roll_Angle : in FLOAT);
    procedure Set_Pitch_Angle(Pitch_Angle : in FLOAT);
    procedure Set_Yaw_Angle(Yaw_Angle : in FLOAT);
    procedure Set_Depth(Depth : in FLOAT);
    procedure Set_Acc_X(Acc_X : in FLOAT);
    procedure Set_Acc_Y(Acc_Y : in FLOAT);
    procedure Set_Acc_Z(Acc_Z : in FLOAT);
    procedure Set_Ang_Velocity_X(Ang_Velocity_X : in FLOAT);
    procedure Set_Ang_Velocity_Y(Ang_Velocity_Y : in FLOAT);

```

---

```

    procedure Set_Ang_Velocity_Z(Ang_Velocity_Z : in FLOAT);

    function Get_Sensor_Data return Sensor_Data_type;

private
    Sensor_Data : Sensor_Data_type;
end Protected_Sensor_Data;

end Protected_Sensor_Data_Package;

--
-- Mode: Ada --
-- Filename      : Protected_Sensor_Data_Package.adb
-- Description    : Protected Object containing most recent sensor readings
-- Author        : David Biddle
-- Created On    : Mon 3rd March 2003
-- Last Modified By: David Biddle
-- Last Modified On: 11-06-03
-- Update Count  : 2
-- Status        : Use with caution

with Ada.Text_IO; use Ada.Text_IO;
with Ada.Float_Text_IO; use Ada.Float_Text_IO;
with Ada.Real_Time; use Ada.Real_Time;

package body Protected_Sensor_Data_Package is

    protected body Protected_Sensor_Data is

        Procedure Set_Roll_Angle(Roll_Angle : in FLOAT) is
        begin
            Sensor_Data(1) := Roll_Angle;
        end Set_Roll_Angle;

        Procedure Set_Pitch_Angle(Pitch_Angle : in FLOAT) is
        begin
            Sensor_Data(2) := Pitch_Angle;
        end Set_Pitch_Angle;

        Procedure Set_Yaw_Angle(Yaw_Angle : in FLOAT) is
        begin
            Sensor_Data(3) := Yaw_Angle;
        end Set_Yaw_Angle;

        Procedure Set_Depth(Depth : in FLOAT) is
        begin
            Sensor_Count := Sensor_Count + 1;
            Sensor_Data(4) := Depth;
        end Set_Depth;
    end Protected_Sensor_Data;
end Protected_Sensor_Data_Package;

```

---

```
end Set_Depth;

Procedure Set_Acc_X(Acc_X : in FLOAT) is
begin
    Sensor_Data(5) := Acc_X;
end Set_Acc_X;

Procedure Set_Acc_Y(Acc_Y : in FLOAT) is
begin
    Sensor_Data(6) := Acc_Y;
end Set_Acc_Y;

Procedure Set_Acc_Z(Acc_Z : in FLOAT) is
begin
    Sensor_Data(7) := Acc_Z;
end Set_Acc_Z;

Procedure Set_Ang_Velocity_X(Ang_Velocity_X : in FLOAT) is
begin
    Sensor_Data(8) := Ang_Velocity_X;
end Set_Ang_Velocity_X;

Procedure Set_Ang_Velocity_Y(Ang_Velocity_Y : in FLOAT) is
begin
    Sensor_Data(9) := Ang_Velocity_Y;
end Set_Ang_Velocity_Y;

Procedure Set_Ang_Velocity_Z(Ang_Velocity_Z : in FLOAT) is
begin
    Sensor_Data(10) := Ang_Velocity_Z;
end Set_Ang_Velocity_Z;

function Get_Sensor_Data return Sensor_Data_type is
begin
    return Sensor_Data;
end Get_Sensor_Data;

end Protected_Sensor_Data;

end Protected_Sensor_Data_Package;
```

---

# Inclinometer Testing Code

---

The Ada code used to test the operation of the inclinometers shown below, includes the following files:

- Test3.adb;
- Protect\_Roll\_Sensor\_Package.ads; and
- Protect\_Roll\_Sensor\_Package.adb.

Almost identical code that was created to get readings from the inclinometer sensor, measuring the pitch angle, has been included in the CD.

```
--                                -*- Mode: Ada -*-
-- Filename      : Test3.adb
-- Description   : Test Serial connection to Inclinometers
-- Author       : David Biddle
-- Created On    : Mon 3rd March 2003
-- Last Modified By: .
-- Last Modified On: .
-- Update Count  : 0
-- Status       : Use with caution

with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
with Protected_Roll_Sensor_Package; use Protected_Roll_Sensor_Package;
with Protected_Pitch_Sensor_Package; use Protected_Pitch_Sensor_Package;
with System.Storage_Elements; use System.Storage_Elements;

procedure Test3 is

    Roll_Position:Integer;
    Pitch_Position:Integer;

begin
    Protected_Roll_Sensor.Get_Roll_Address;
    Roll_Position := Protected_Roll_Sensor.Get_Roll_Position;
    Put(Roll_Position);
```

---

```

    New_Line;
    Roll_Position := Protected_Roll_Sensor.Get_Roll_Position;
    Put(Roll_Position);
    New_Line;
    Roll_Position := Protected_Roll_Sensor.Get_Roll_Position;
    Put(Roll_Position);
    New_Line;

    Protected_Pitch_Sensor.Get_Pitch_Address;
    Pitch_Position := Protected_Pitch_Sensor.Get_Pitch_Position;
    Put(Pitch_Position);
    New_Line;
    Pitch_Position := Protected_Pitch_Sensor.Get_Pitch_Position;
    Put(Pitch_Position);
    New_Line;
    Pitch_Position := Protected_Pitch_Sensor.Get_Pitch_Position;
    Put(Pitch_Position);
    New_Line;
end Test3;

--                                     -*- Mode: Ada -*-
-- Filename      : Protected_Roll_Sensor_Package.ads
-- Description    : Serial connection to InclInometers
-- Author        : David Biddle
-- Created On    : Mon 3rd March 2003
-- Last Modified By: David Biddle
-- Last Modified On: .
-- Update Count  : 0
-- Status        : Use with care

with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;

package Protected_Roll_Sensor_Package is

    protected Protected_Roll_Sensor is

        procedure Get_Roll_Address;
        function Get_Roll_Position return Integer;

    end Protected_Roll_Sensor;

end Protected_Roll_Sensor_Package;

--                                     -*- Mode: Ada -*-
-- Filename      : Protected_Roll_Sensor_Package.adb
-- Description    : Serial connection to InclInometer measuring Roll Angle
-- Author        : David Biddle
-- Created On    : Mon 3rd March 2003

```

---

```

-- Last Modified By: David Biddle
-- Last Modified On:
-- Update Count      : 0
-- Status             : Use with caution

with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
with Serial_Ports; use Serial_Ports;
with System.Storage_Elements; use System.Storage_Elements;

package body Protected_Roll_Sensor_Package is
  Port   : Serial_Ports.Serial_Port;
  Device : constant String := "/dev/ttyS1";
  Roll_Address : Integer:=0;
  Data   : Storage_Array(1..1000);
  Last   : Storage_Offset;

  protected body Protected_Roll_Sensor is

    procedure Get_Roll_Address is
      -- See A2 Communications Protocol for Inclinator protocol.

      Multiple_Byte_Request_CMD : constant Storage_Array := (1 => 2#1111_1111#);

      Read_Roll_CMD : constant Storage_Array := (1 => 2#0000_0110#,
                                                  2 => 2#0000_0000#,
                                                  3 => 2#0000_0000#,
                                                  4 => 35,
                                                  5 => 78);

    begin

      Open_Serial_Port(Port, Device);
      Setup_Serial_Port(Port, Baud_9_600);
      delay 0.02;
      Write(Port, Multiple_Byte_Request_CMD);
      delay 0.02;
      Write(Port, Read_Roll_CMD);

      Read(Port, Data, Last);
      delay 0.02;
      Read(Port, Data, Last);
      Roll_Address := Integer(Data(6));

    end Get_Roll_Address;

    function Get_Roll_Position return Integer is

      Roll_Position_CMD : Storage_Array := (1 => 2#0001_0000#);

```



---

```
    Rotation_Sum    : Integer:=0;

begin
    Roll_Position_CMD(1) := Storage_Element(Integer(Roll_Position_CMD(1)) + Roll_Address);
    Write(Port, Roll_Position_CMD);
    delay 0.01;
    Read(Port, Data, Last);
    for i in Data'First..Last loop
        if i /= Data'First then
            Rotation_Sum := Rotation_Sum*256 + Integer(Data(i));
        end if;
    end loop;

    return Rotation_Sum;

end Get_Roll_Position;

end Protected_Roll_Sensor;

end Protected_Roll_Sensor_Package;
```

---

## CD-ROM directory Structure

---

There are 3 directories in the CD-ROM. These are as follows:

- Thesis - contains all the Latex files used to reproduce this thesis
- Ada - The Ada Code for the implementation of the controller and the testing of the inclinometers is included.
- Amplifiers - The files used to encode the Accelus Amplifiers.